Inheritance

Objectives

- Study concepts: superclass, subclass
- Functions in inheritance Understand the "is-a" relationship
- Overloading and Overriding

Inheritance

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

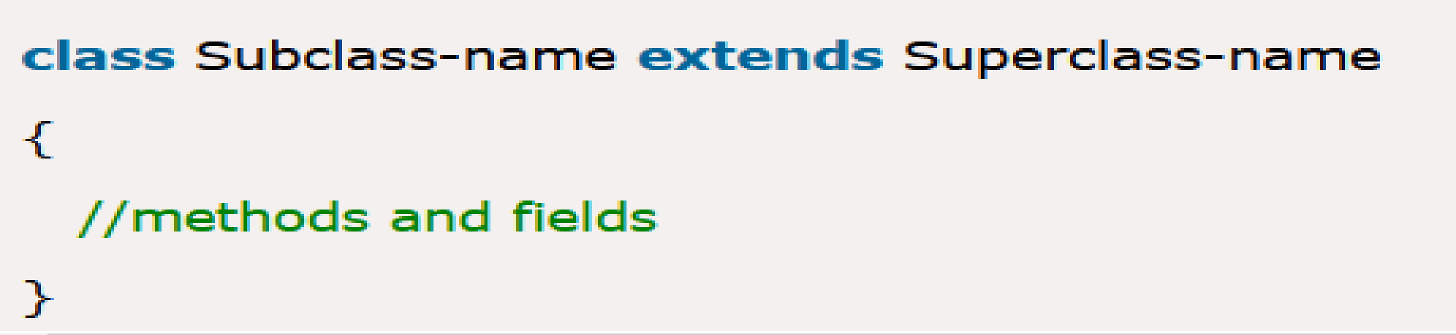
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

- Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Inheritance

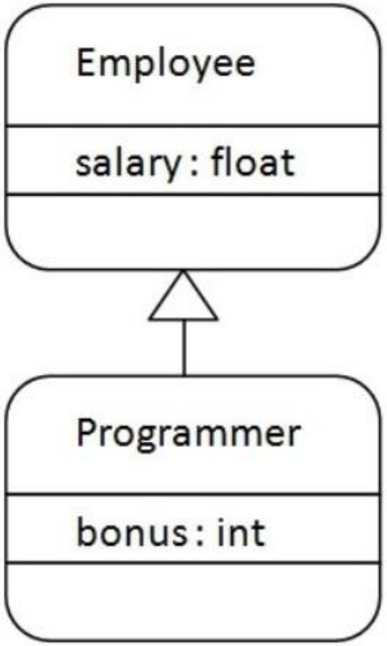
- There are some sub-classes from one super class ➔ An inheritance is a relationship where objects share a common structure: the structure of one object is a sub-structure of another object.
- The extends keyword is used to create sub-class.
- A class can be directly derived from only one class (Java is a single-inherited OOP language).
- If a class does not have any superclass, then it is implicitly derived from Object class.
- Unlike other members, constructor cannot be inherited (constructor of super class can not initialize sub-class objects)

The syntax of Java Inheritance

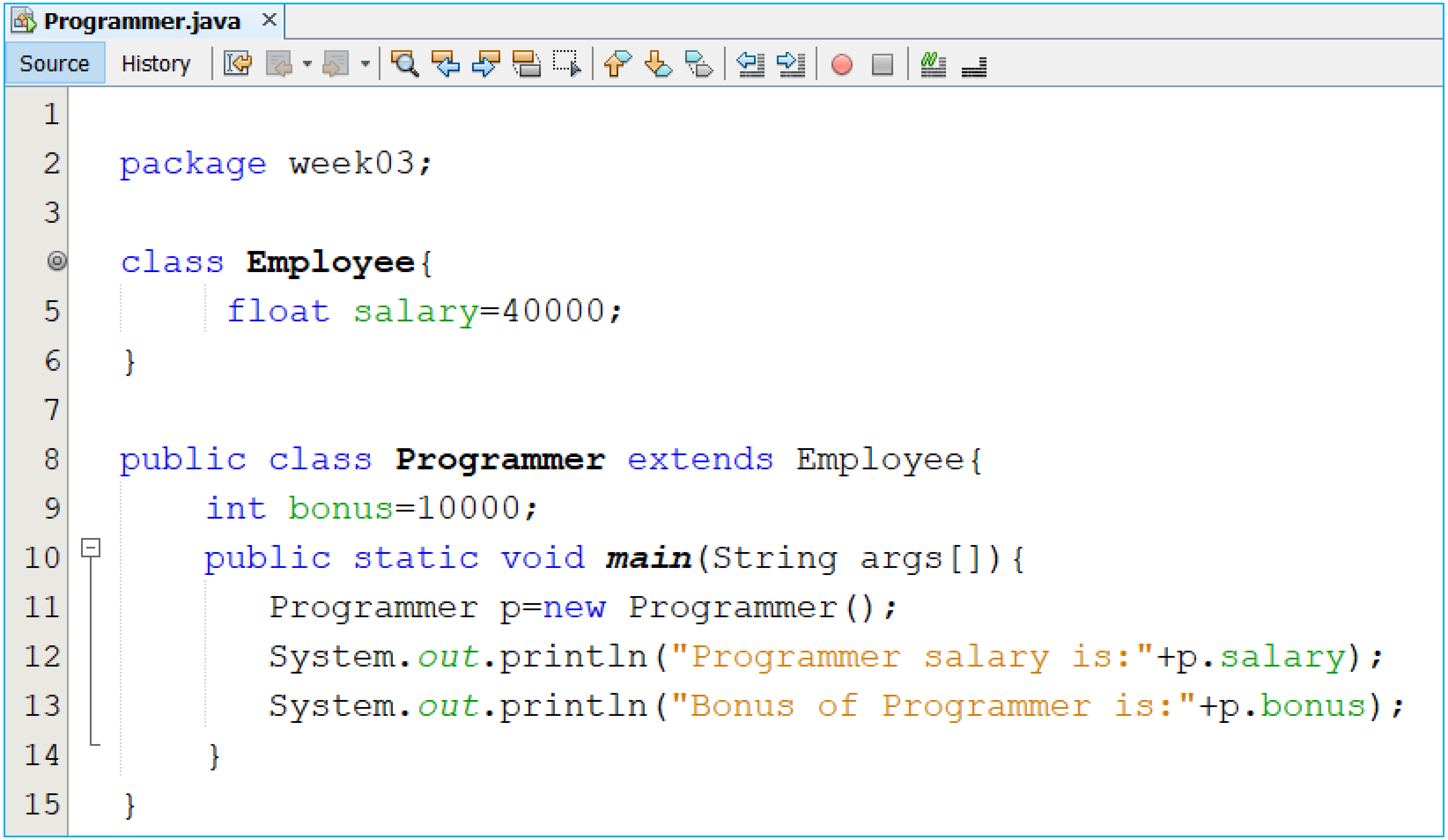


```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

Java Inheritance Example

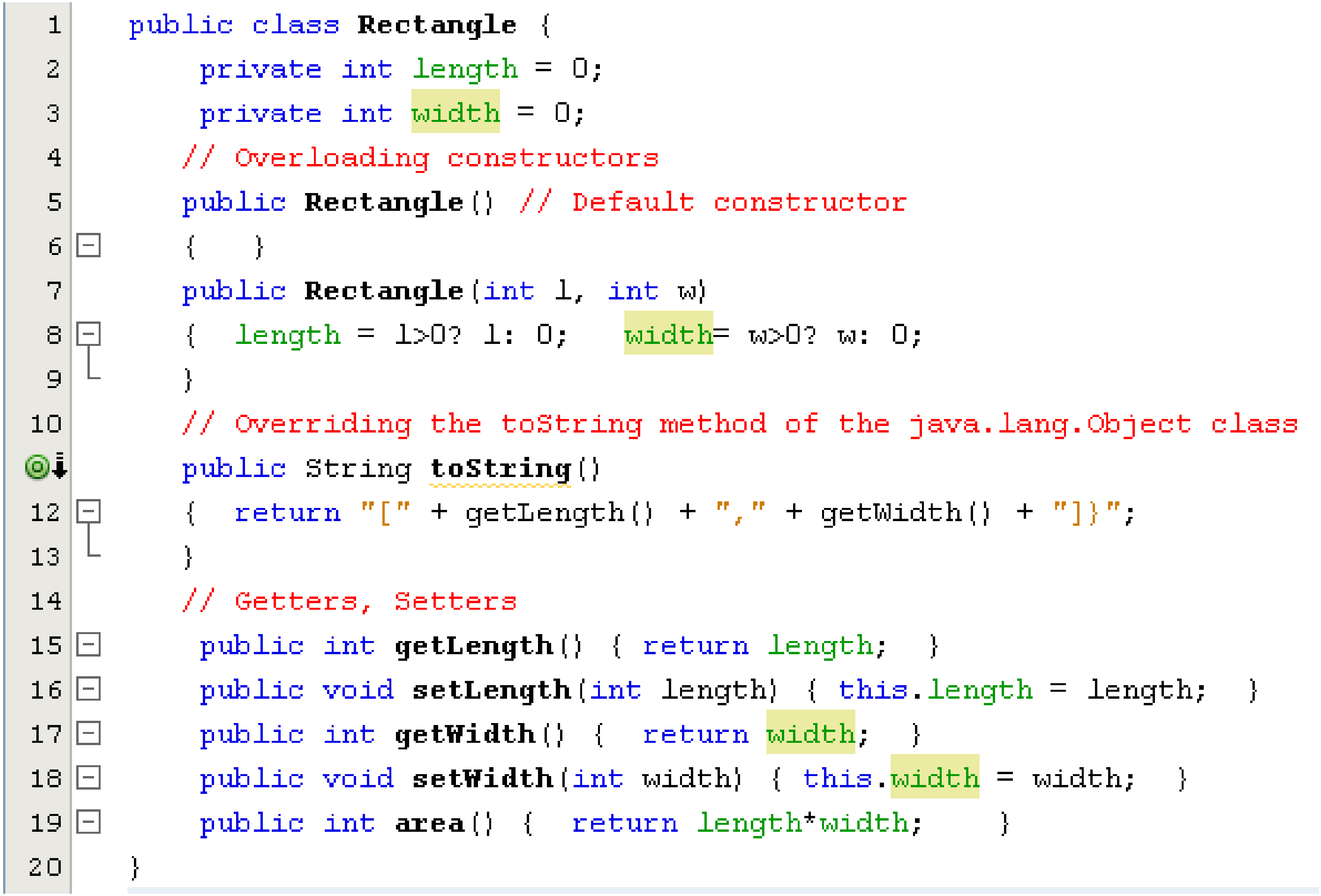


Java Inheritance Example



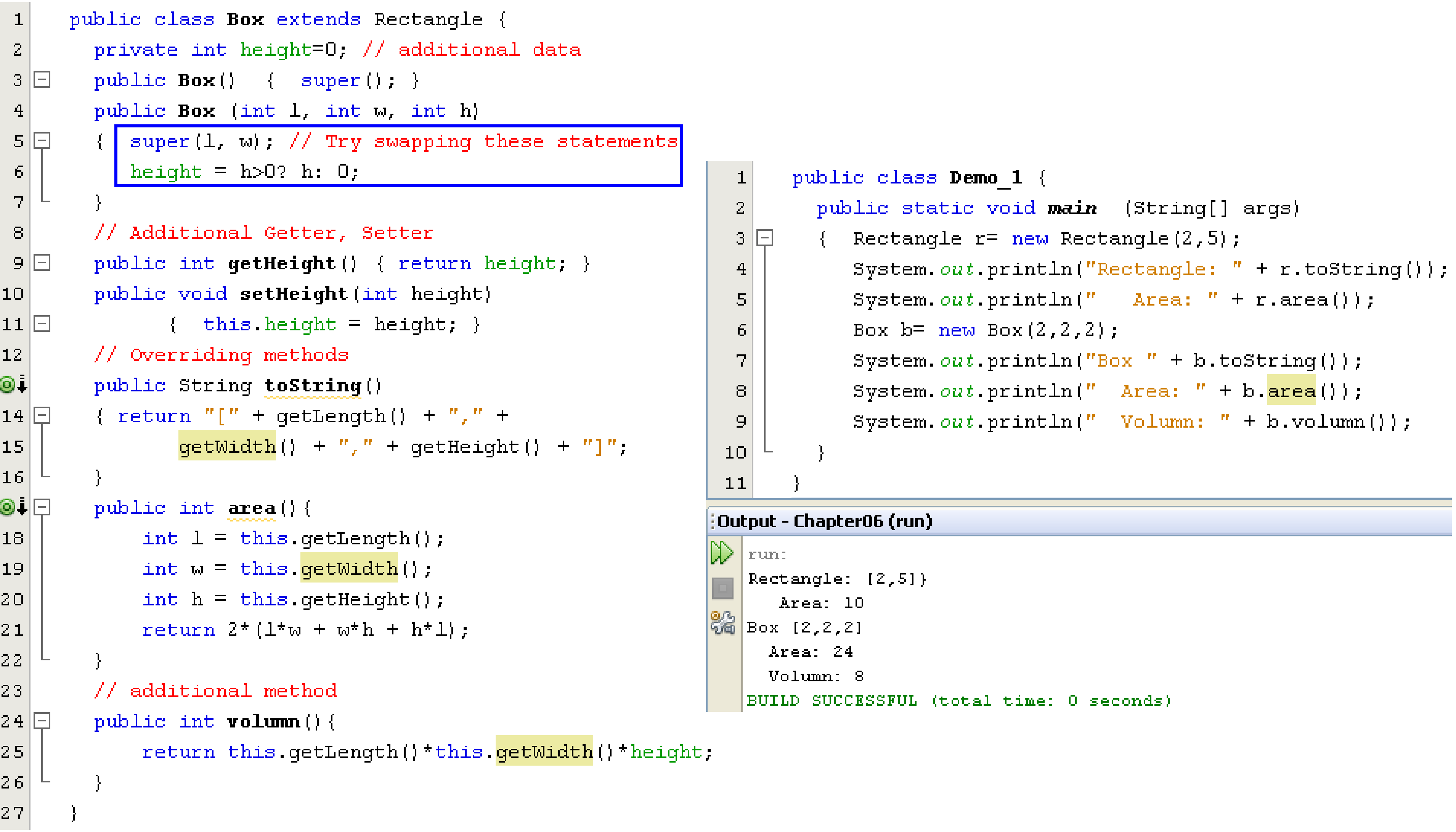
```java

package week03;

class Employee{
        float salary=40000;
}

public class Programmer extends Employee{
    int bonus=10000;
    public static void main(String args[]){
       Programmer p=new Programmer();
       System.out.println("Programmer salary is:"+p.salary);
       System.out.println("Bonus of Programmer is:"+p.bonus);
    }
}
```

Inheritance...



```java
public class Rectangle {
    private int length = 0;
    private int width = 0;
   // Overloading constructors
   public Rectangle() // Default constructor
   {    }
   public Rectangle(int l, int w)
   {  length = l>0? l: 0;   width= w>0? w: 0;
   }
   // Overriding the toString method of the java.lang.Object class
   public String toString()
   {  return "[" + getLength() + "," + getWidth() + "]}";
   }
   // Getters, Setters
    public int getLength() { return length;  }
    public void setLength(int length) { this.length = length;  }
    public int getWidth() {  return width;  }
    public void setWidth(int width) { this.width = width;  }
    public int area() {  return length*width;     }
}
```

Inheritance…



```java
public class Box extends Rectangle {
  private int height=0; // additional data
  public Box()  {  super(); }
  public Box (int l, int w, int h)
  { super(l, w); // Try swapping these statements
    height = h>0? h: 0;
  }
  // Additional Getter, Setter
  public int getHeight() { return height; }
  public void setHeight(int height)
        {  this.height = height; }
  // Overriding methods
  public String toString()
  { return "[" + getLength() + "," +
         getWidth() + "," + getHeight() + "]";
  }
  public int area(){
      int l = this.getLength();
      int w = this.getWidth();
      int h = this.getHeight();
      return 2*(l*w + w*h + h*l);
  }
  // additional method
  public int volumn(){
      return this.getLength()*this.getWidth()*height;
  }
}
```

```java
public class Demo_1 {
  public static void main  (String[] args)
  {  Rectangle r= new Rectangle(2,5);
     System.out.println("Rectangle: " + r.toString());
     System.out.println("   Area: " + r.area());
     Box b= new Box(2,2,2);
     System.out.println("Box " + b.toString());
     System.out.println("  Area: " + b.area());
     System.out.println("  Volumn: " + b.volumn());
  }
}
```

Output - Chapter06 (run)

```
run:
Rectangle: [2,5])
   Area: 10
Box [2,2,2]
  Area: 24
  Volumn: 8
BUILD SUCCESSFUL (total time: 0 seconds)
```

Inheritance…: “super” Keyword

- Constructors Are **Not** Inherited
- super(...) for Constructor Reuse
  - super(arguments); *//invoke a superclass constructor*
  - The call ***must* be the *first* statement in the subclass constructor**
- Replacing the Default Parameterless Constructor

Inheritance…: “super” Keyword

- We use the Java keyword super as the qualifier for a method call:
  super. methodName(arguments);
- Whenever we wish to invoke the version of method methodName that was defined by our superclass.
- **super()** is used to access the superclass's constructor. And It must be the first statement in the constructor of the subclass.

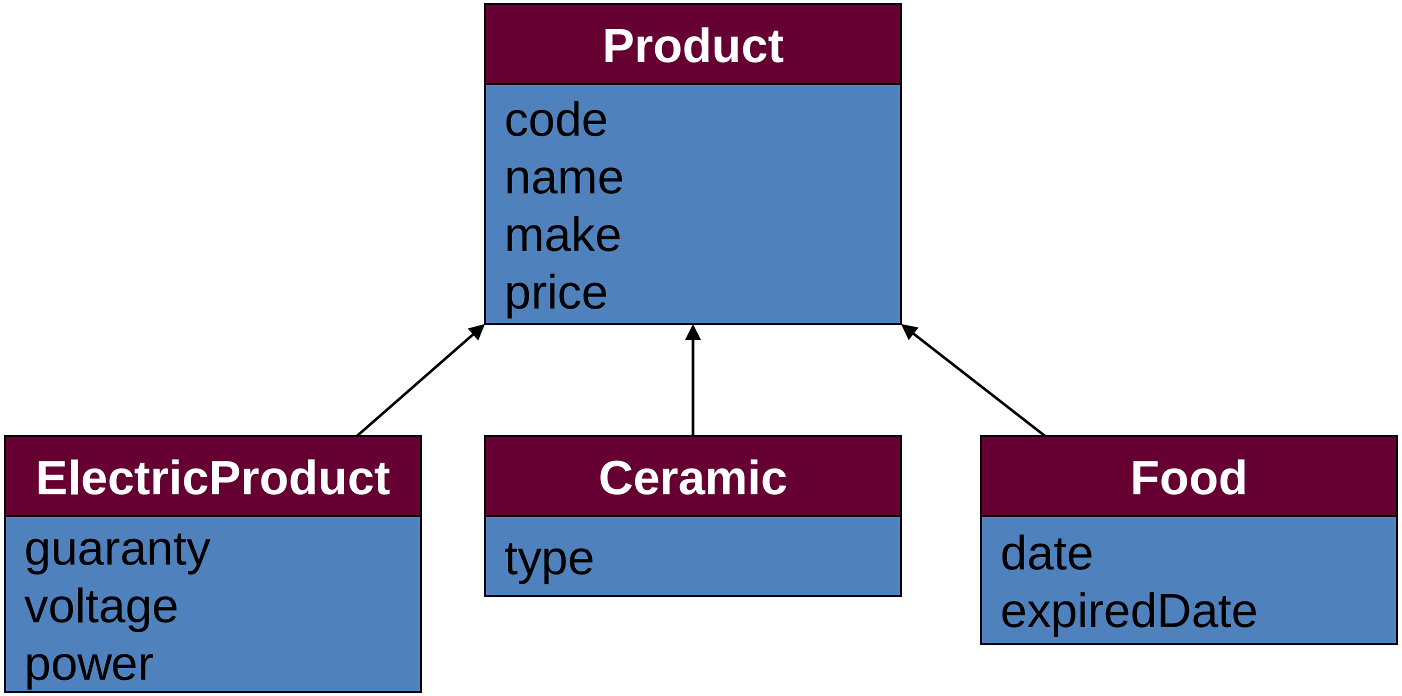
Common Modifiers

super: Keyword for calling a member declared in the father class.
If contructor of sub-class calls a constructor of it's father using super, it must be the first statement in the sub-class constructor.

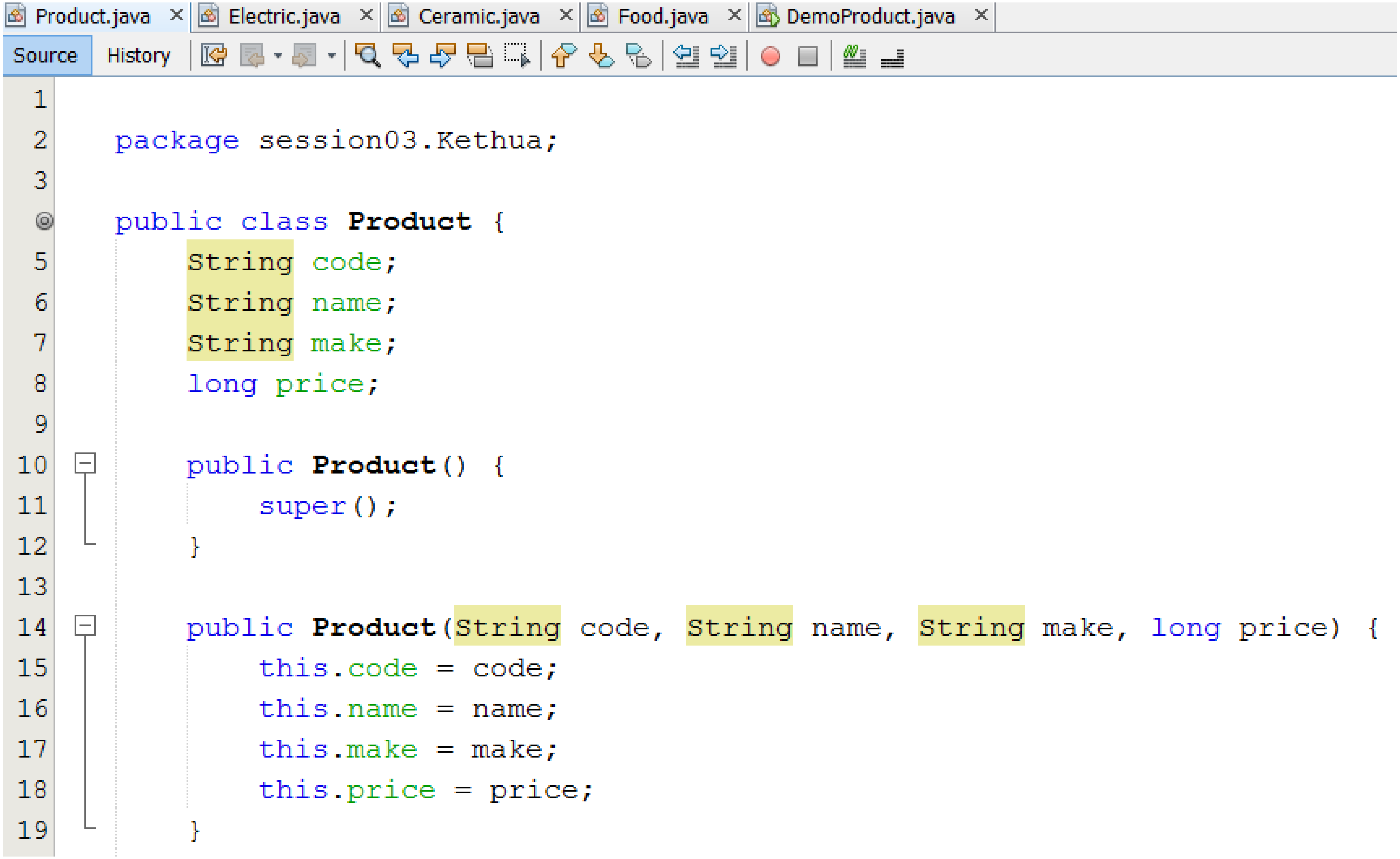
Inheritance…

- **How to construct a class hierarchy? → Intersection**

- Electric Products< code, name, make, price, guaranty, voltage, power>
- Ceramic Products < code, name, make, price, type >
- Food Products < code, name, make, price , date, expiredDate >

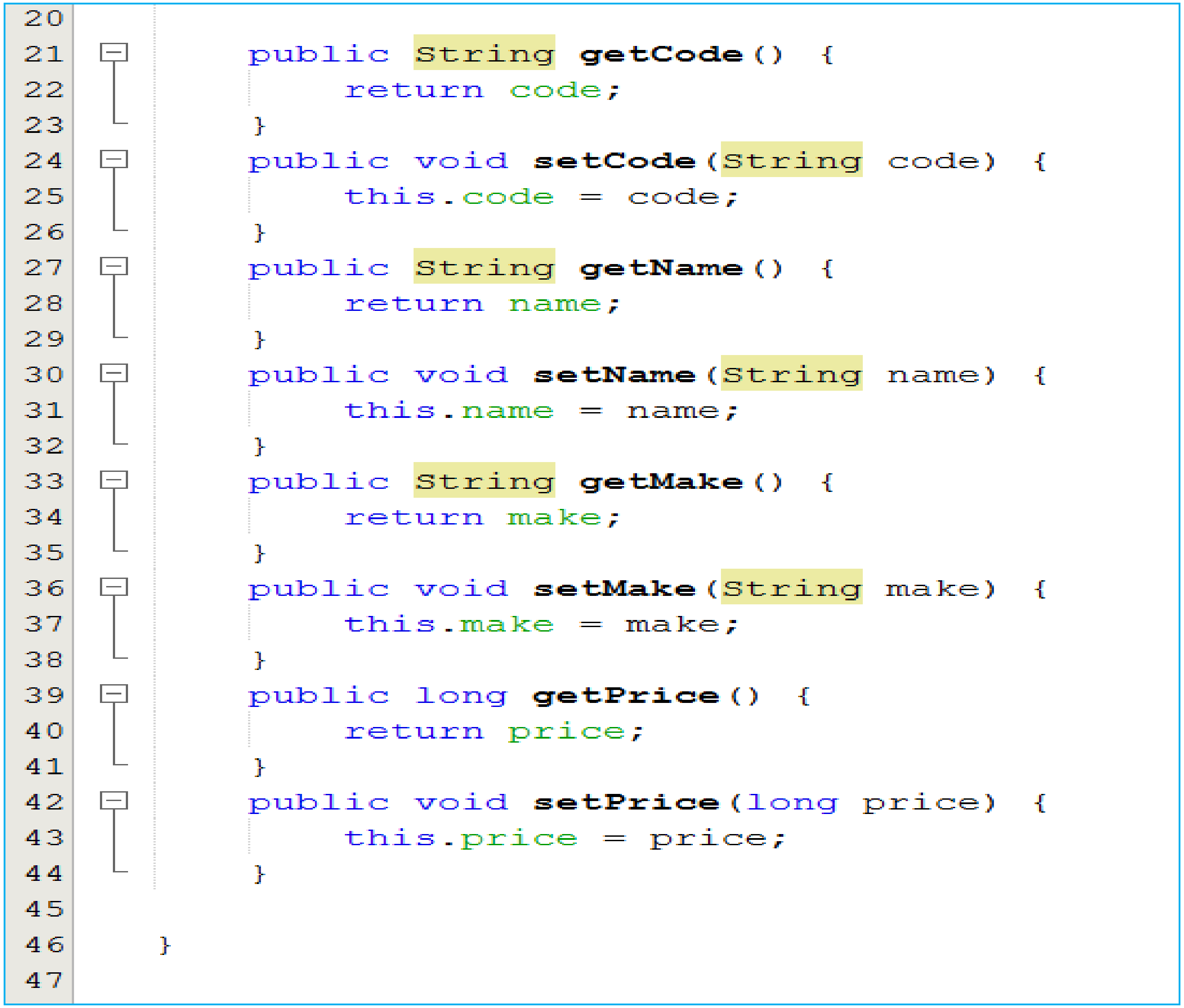


Demo Inheritance



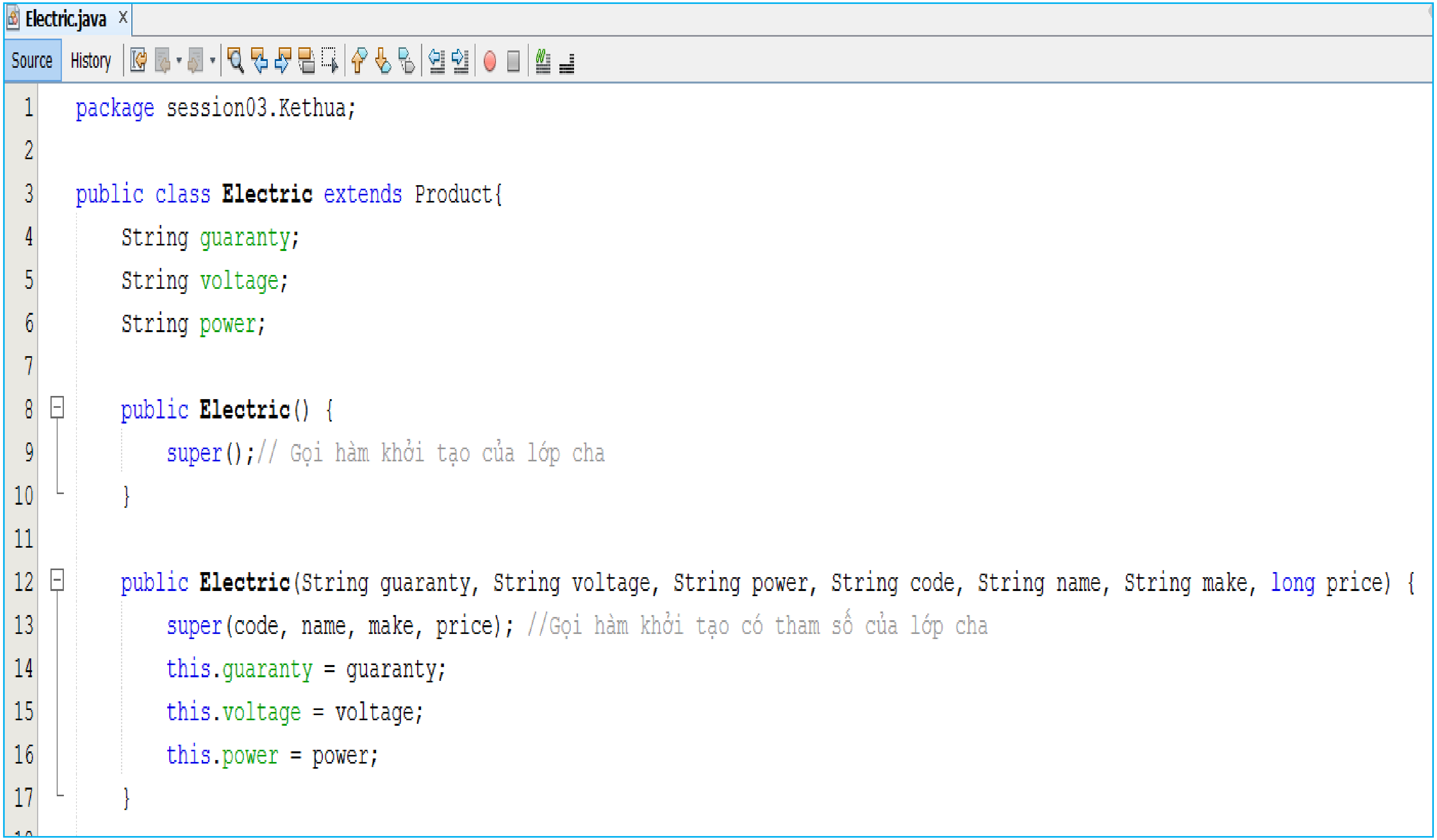
```java

package session03.Kethua;

public class Product {
    String code;
    String name;
    String make;
    long price;

    public Product() {
        super();
    }

    public Product(String code, String name, String make, long price) {
        this.code = code;
        this.name = name;
        this.make = make;
        this.price = price;
    }
```

Demo Inheritance



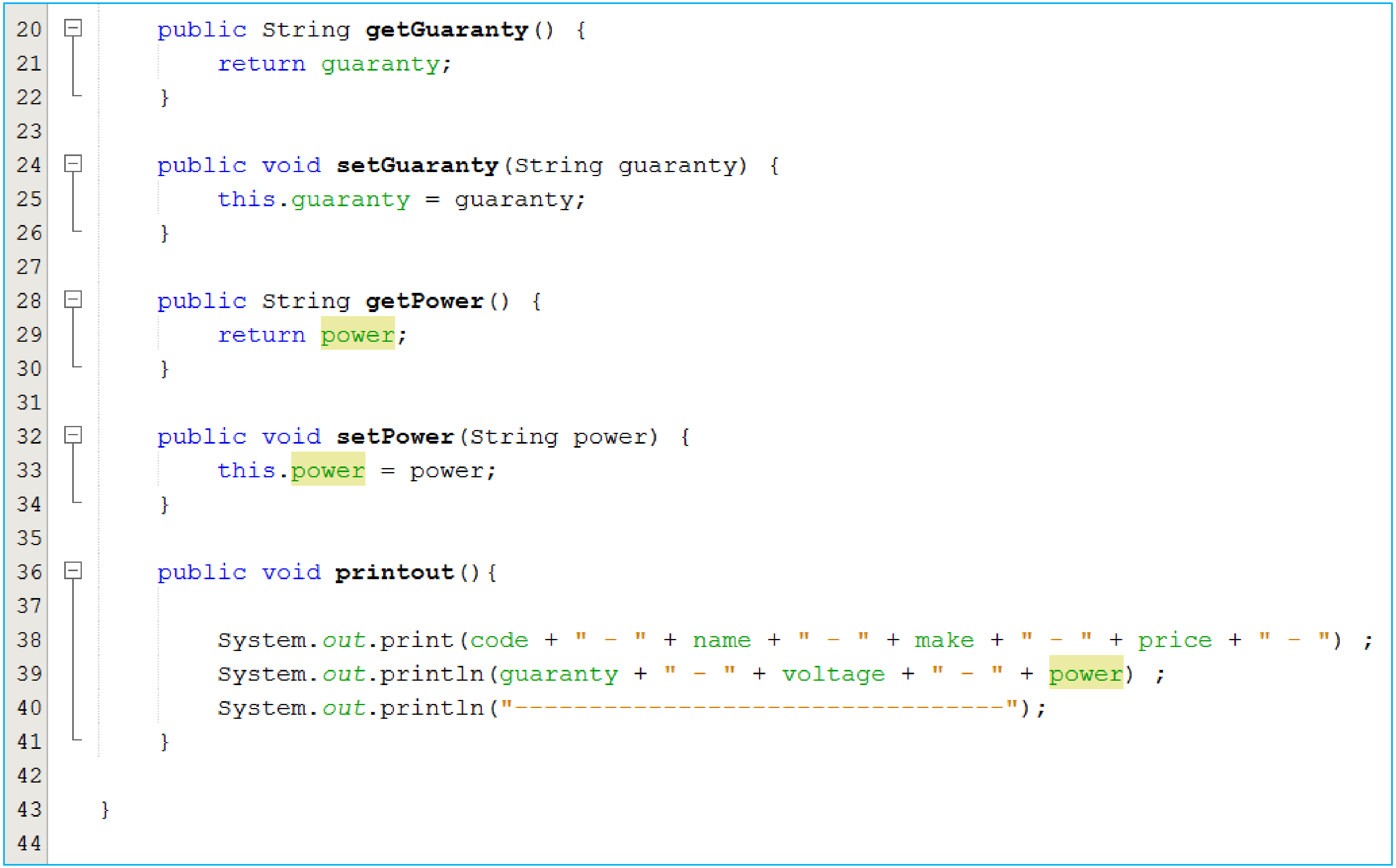
```
    public String getCode() {
        return code;
    }
    public void setCode(String code) {
        this.code = code;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getMake() {
        return make;
    }
    public void setMake(String make) {
        this.make = make;
    }
    public long getPrice() {
        return price;
    }
    public void setPrice(long price) {
        this.price = price;
    }

}
```

Demo Inheritance



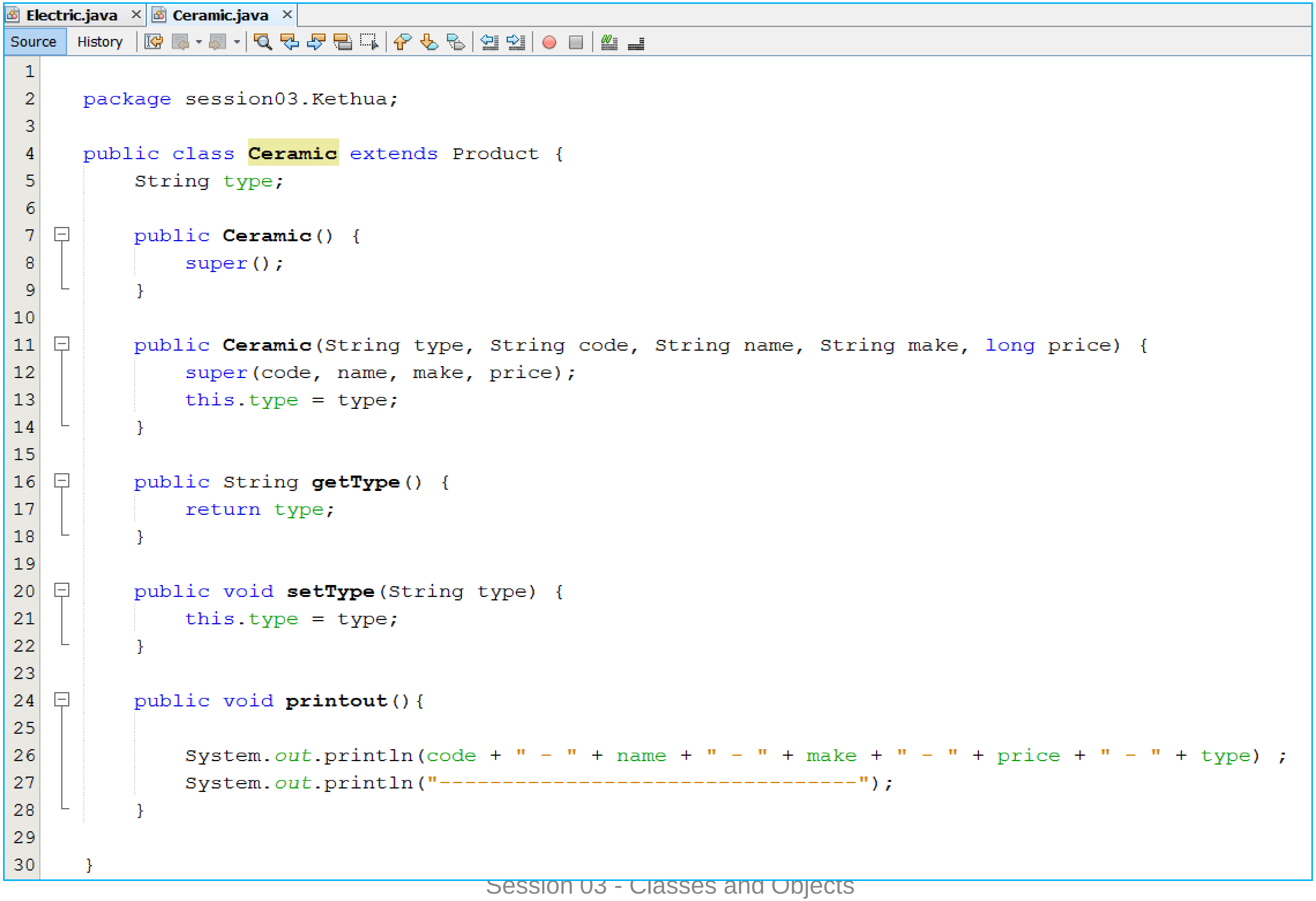
```
package session03.Kethua;

public class Electric extends Product{
    String guaranty;
    String voltage;
    String power;

    public Electric() {
        super();// Gọi hàm khởi tạo của lớp cha
    }

    public Electric(String guaranty, String voltage, String power, String code, String name, String make, long price) {
        super(code, name, make, price); //Gọi hàm khởi tạo có tham số của lớp cha
        this.guaranty = guaranty;
        this.voltage = voltage;
        this.power = power;
    }
```

Demo Inheritance



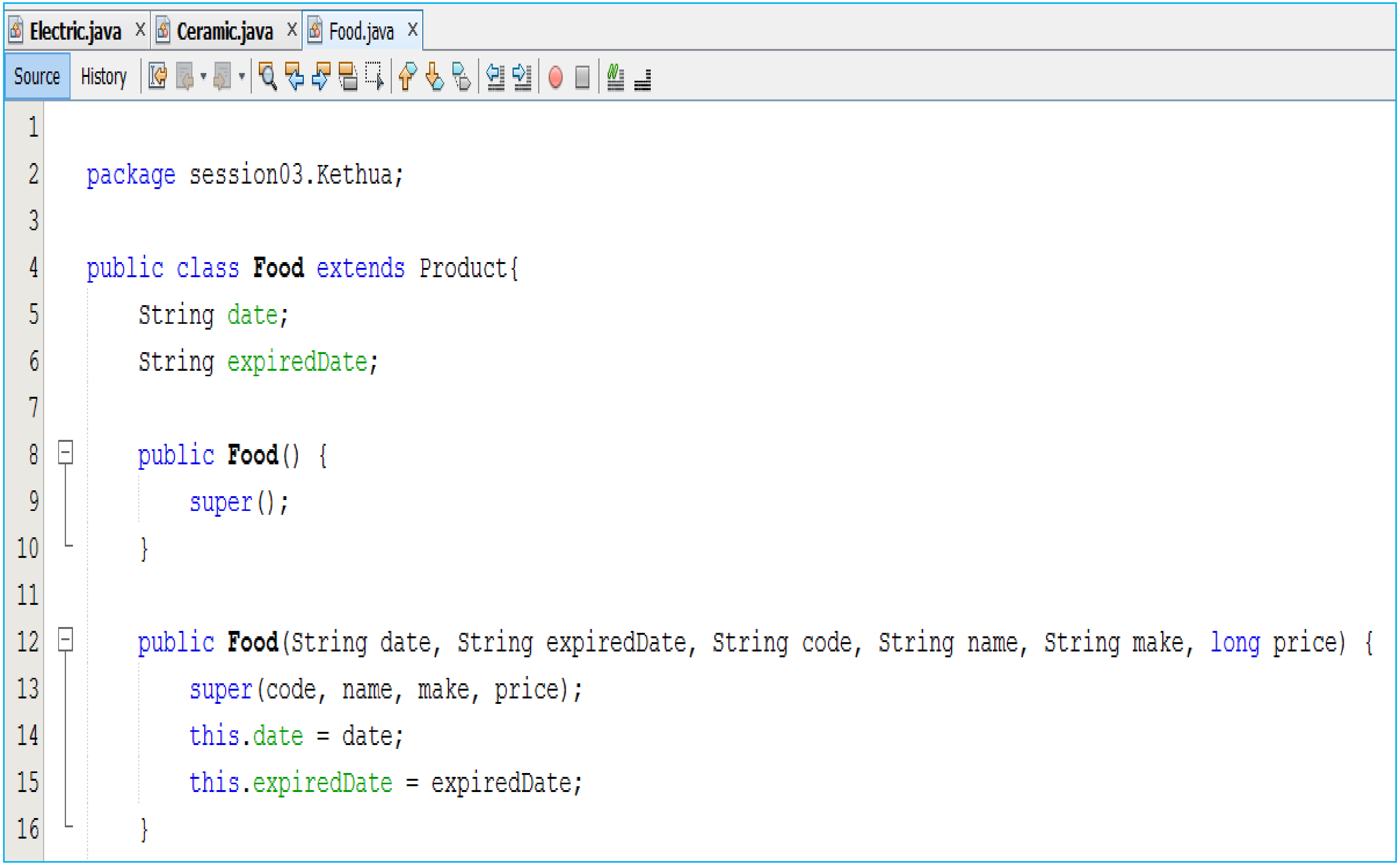
```
    public String getGuaranty() {
        return guaranty;
    }

    public void setGuaranty(String guaranty) {
        this.guaranty = guaranty;
    }

    public String getPower() {
        return power;
    }

    public void setPower(String power) {
        this.power = power;
    }

    public void printout(){

        System.out.print(code + " - " + name + " - " + make + " - " + price + " - ") ;
        System.out.println(guaranty + " - " + voltage + " - " + power) ;
        System.out.println("---------------------------------");
    }

}

```

Demo Inheritance



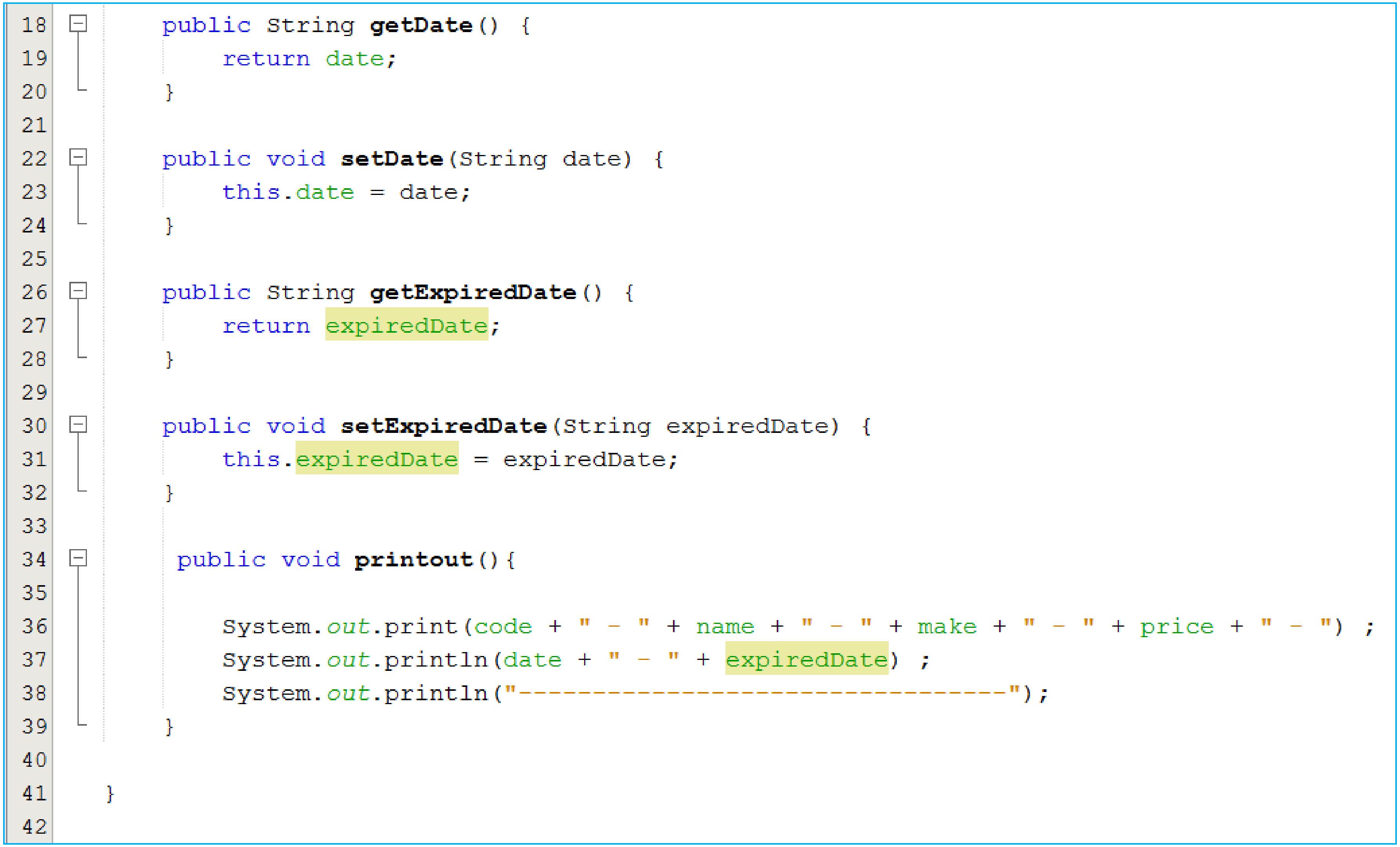
```java
package session03.Kethua;

public class Ceramic extends Product {
    String type;

    public Ceramic() {
        super();
    }

    public Ceramic(String type, String code, String name, String make, long price) {
        super(code, name, make, price);
        this.type = type;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public void printout(){

        System.out.println(code + " - " + name + " - " + make + " - " + price + " - " + type) ;
        System.out.println("---------------------------------");
    }

}
```

Demo Inheritance



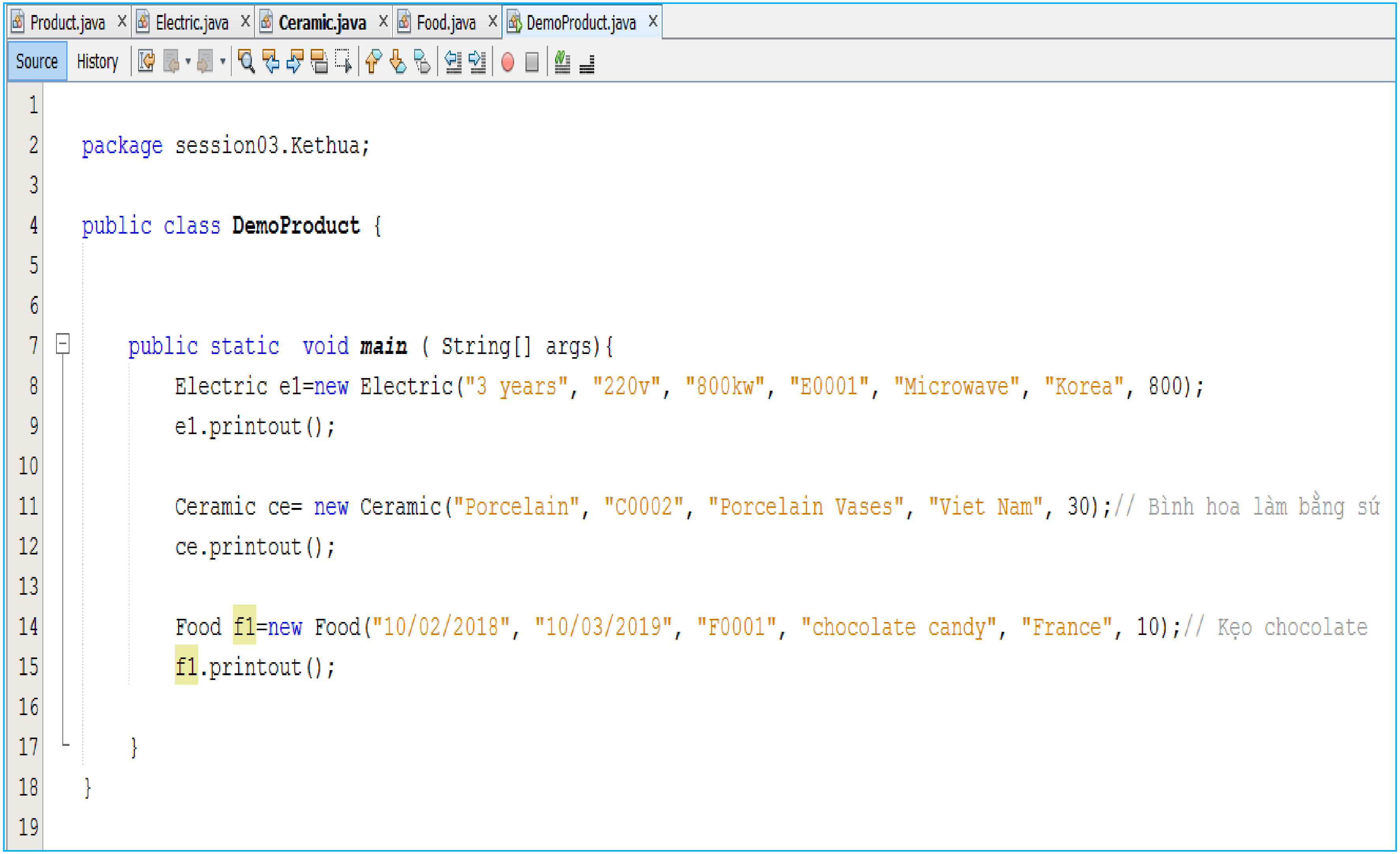
```java

package session03.Kethua;

public class Food extends Product{
    String date;
    String expiredDate;

    public Food() {
        super();
    }

    public Food(String date, String expiredDate, String code, String name, String make, long price) {
        super(code, name, make, price);
        this.date = date;
        this.expiredDate = expiredDate;
    }
```

Demo Inheritance



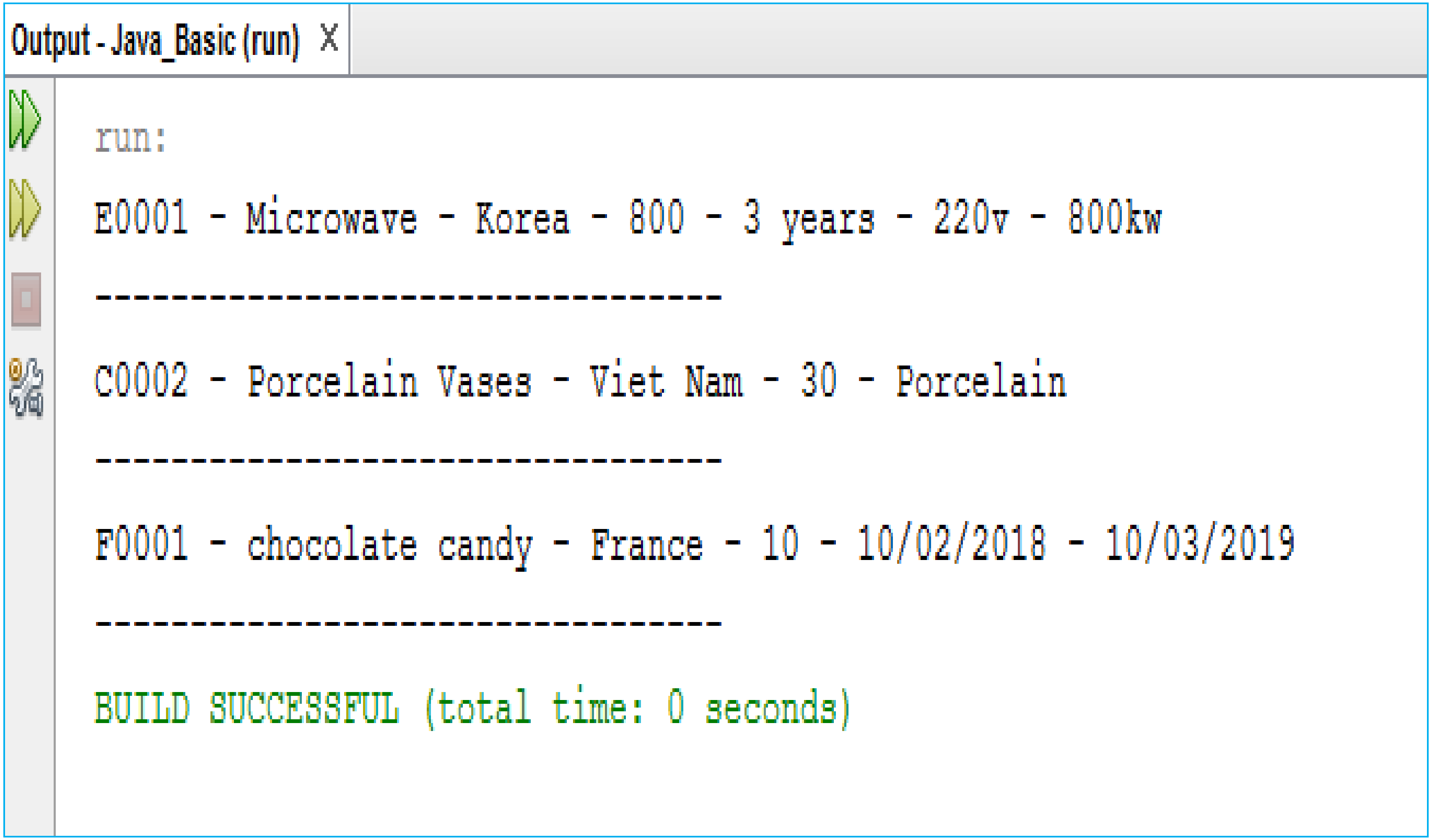
```
    public String getDate() {
        return date;
    }

    public void setDate(String date) {
        this.date = date;
    }

    public String getExpiredDate() {
        return expiredDate;
    }

    public void setExpiredDate(String expiredDate) {
        this.expiredDate = expiredDate;
    }

     public void printout(){

        System.out.print(code + " - " + name + " - " + make + " - " + price + " - ") ;
        System.out.println(date + " - " + expiredDate) ;
        System.out.println("---------------------------------");
    }

}
```

Demo Inheritance



```
package session03.Kethua;

public class DemoProduct {


    public static  void main ( String[] args){
        Electric e1=new Electric("3 years", "220v", "800kw", "E0001", "Microwave", "Korea", 800);
        e1.printout();

        Ceramic ce= new Ceramic("Porcelain", "C0002", "Porcelain Vases", "Viet Nam", 30);// Bình hoa làm bằng sứ
        ce.printout();

        Food f1=new Food("10/02/2018", "10/03/2019", "F0001", "chocolate candy", "France", 10);// Kẹo chocolate
        f1.printout();

    }
}

```

Demo Inheritance



```
run:
E0001 - Microwave - Korea - 800 - 3 years - 220v - 800kw
-----------------------------------
C0002 - Porcelain Vases - Viet Nam - 30 - Porcelain
-----------------------------------
F0001 - chocolate candy - France - 10 - 10/02/2018 - 10/03/2019
-----------------------------------
BUILD SUCCESSFUL (total time: 0 seconds)
```

Implementing Object-Oriented Relationships

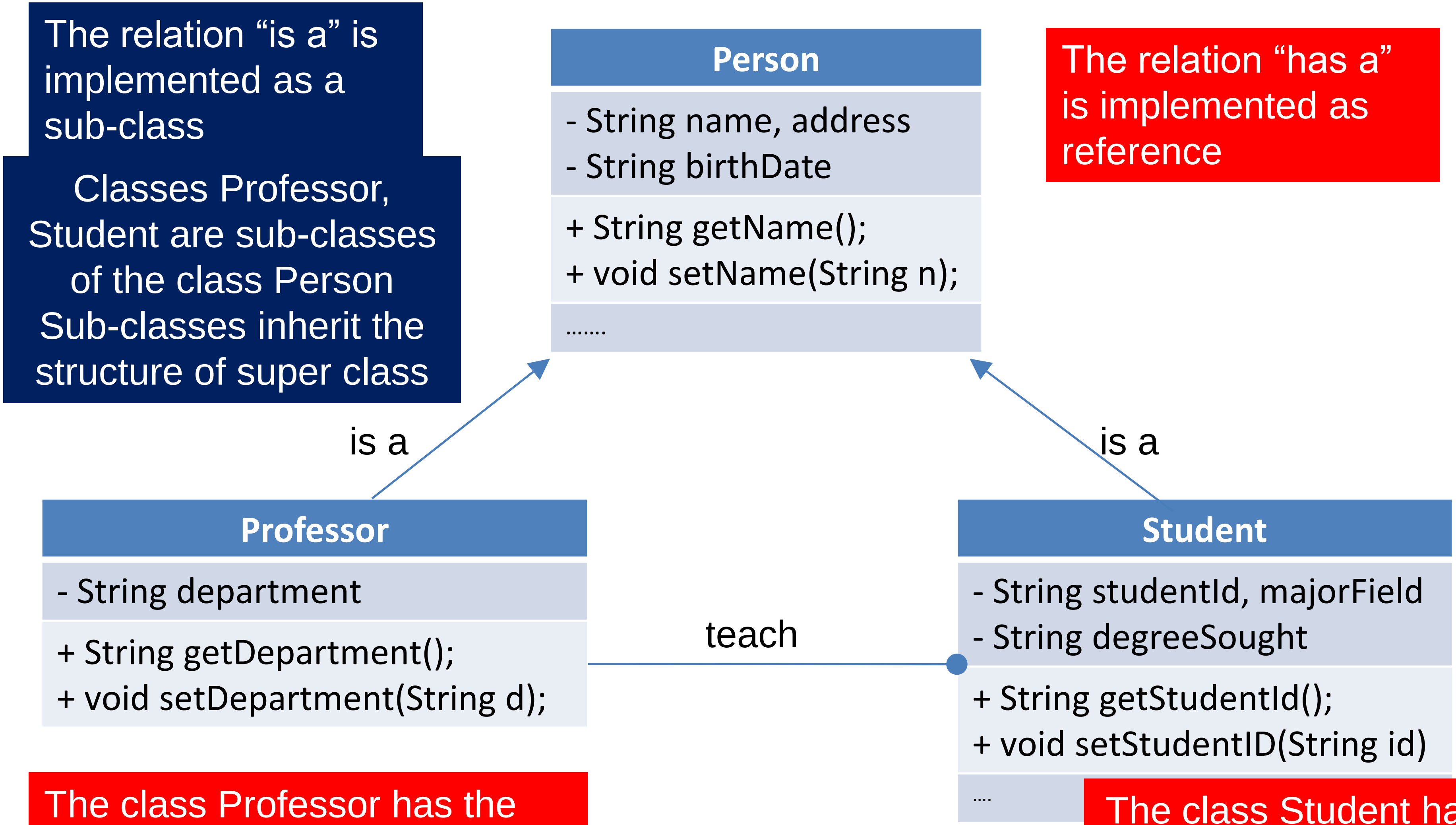
- 3 common relations in classes:
  - “is a/ a kind of”
  - “has a”
  - association
- Examples:
  - Student is a person
  - “A home is a house that has a family and a pet.”
  - An invoice contains some products and a product can be contained in some invoices

Implementing Object-Oriented Relationships…

The relation “is a” is implemented as a sub-class

Classes Professor, Student are sub-classes of the class Person
Sub-classes inherit the structure of super class

The relation “has a” is implemented as reference



Person

- String name, address
- String birthDate

+ String getName();
+ void setName(String n);

…….

is a

is a

Professor

- String department

+ String getDepartment();
+ void setDepartment(String d);

teach

Student

- String studentId, majorField
- String degreeSought

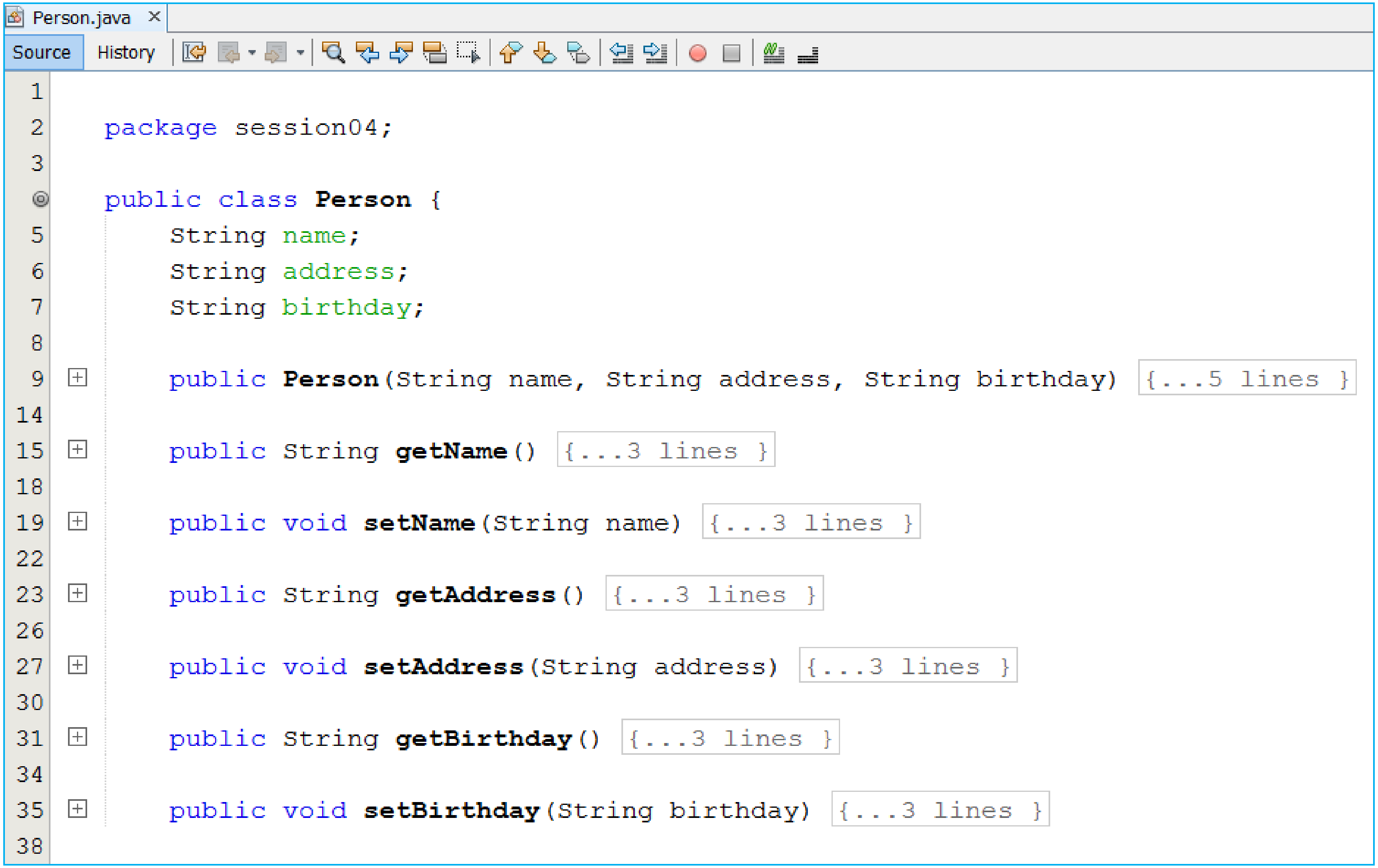
+ String getStudentId();
+ void setStudentID(String id)

….

The class Professor has the field Student[] students

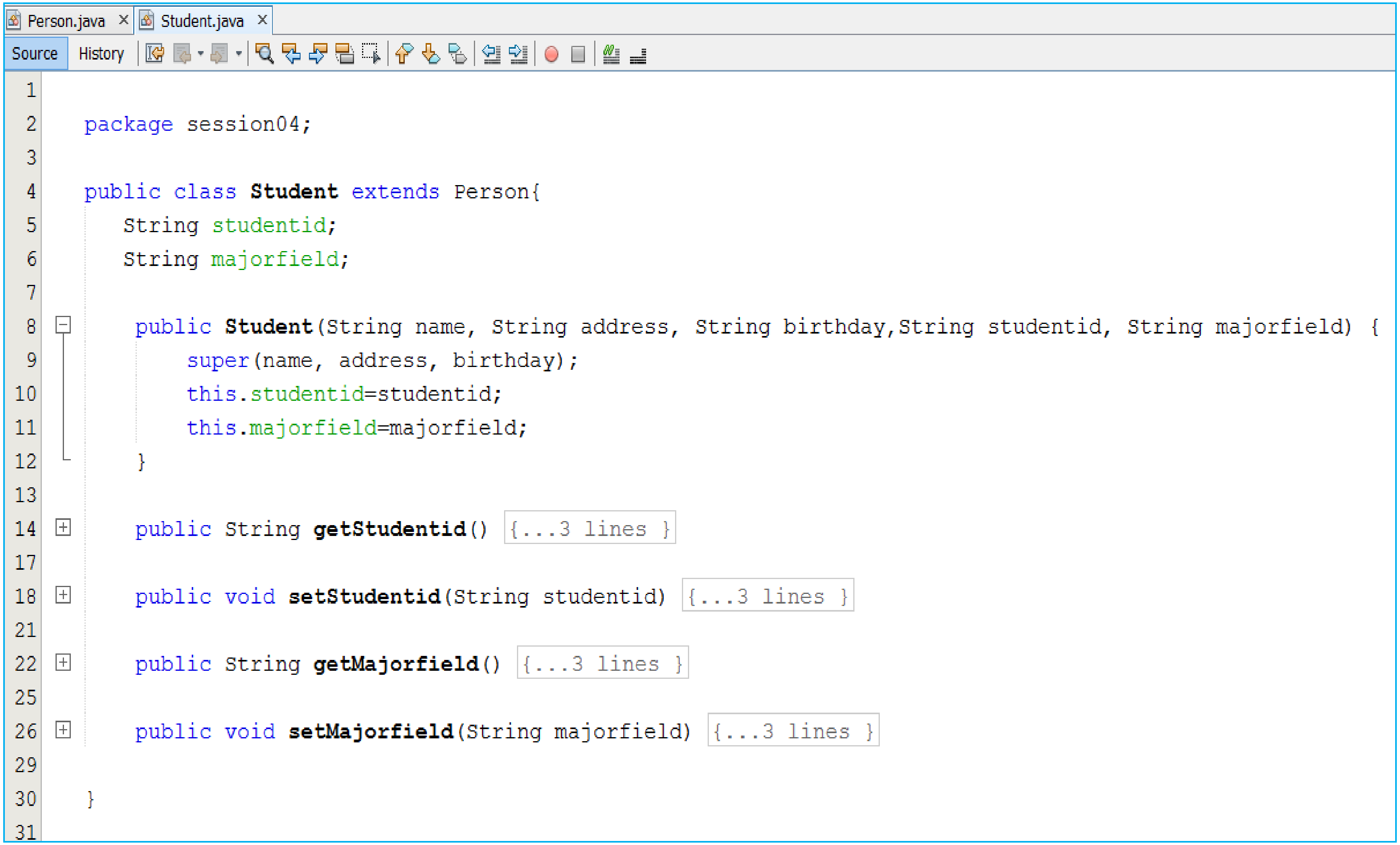
The class Student has the field Professor pr

Demo Object-Oriented Relationships…



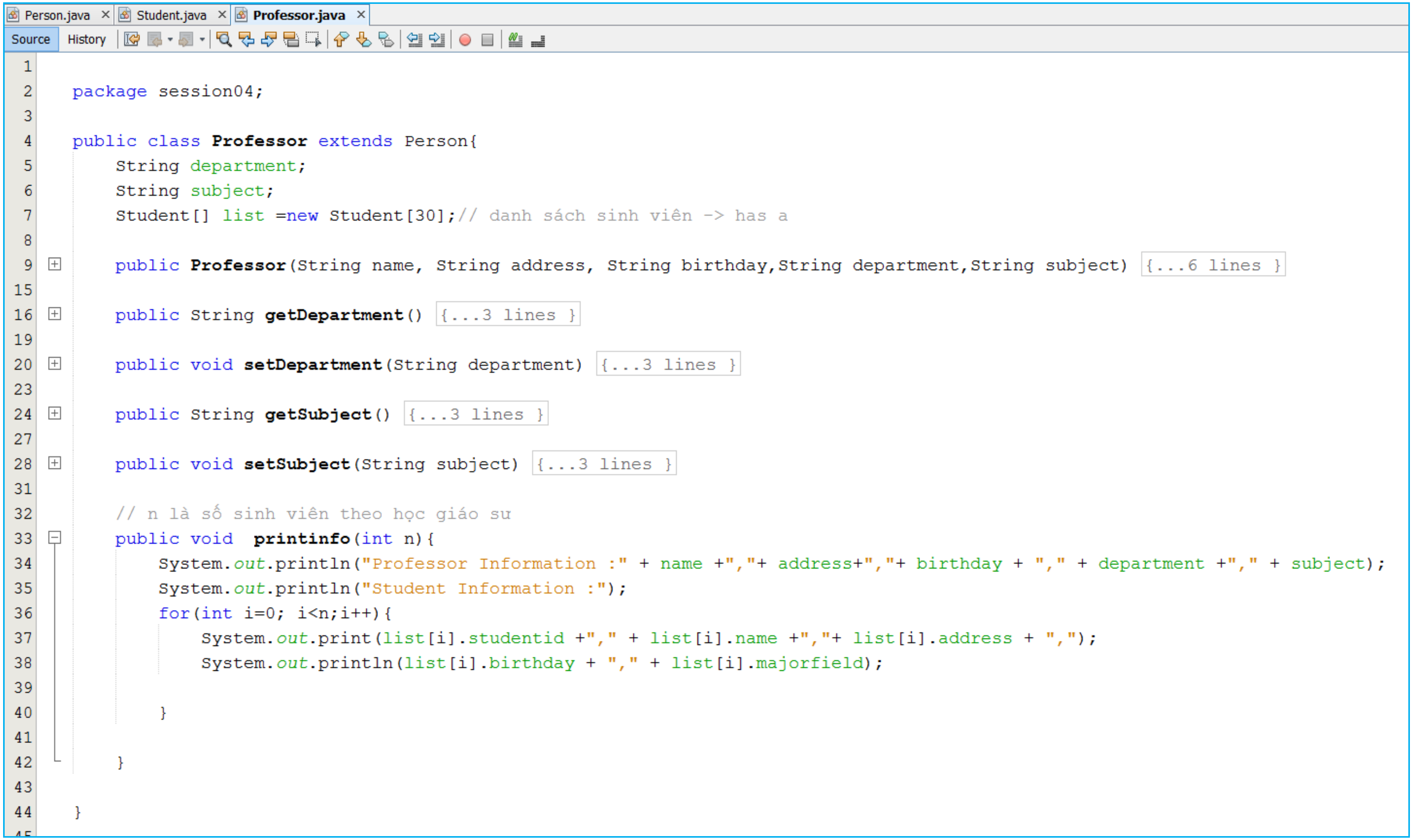
```

package session04;

public class Person {
    String name;
    String address;
    String birthday;

    public Person(String name, String address, String birthday) {...5 lines }

    public String getName() {...3 lines }

    public void setName(String name) {...3 lines }

    public String getAddress() {...3 lines }

    public void setAddress(String address) {...3 lines }

    public String getBirthday() {...3 lines }

    public void setBirthday(String birthday) {...3 lines }

```

Demo Object-Oriented Relationships…



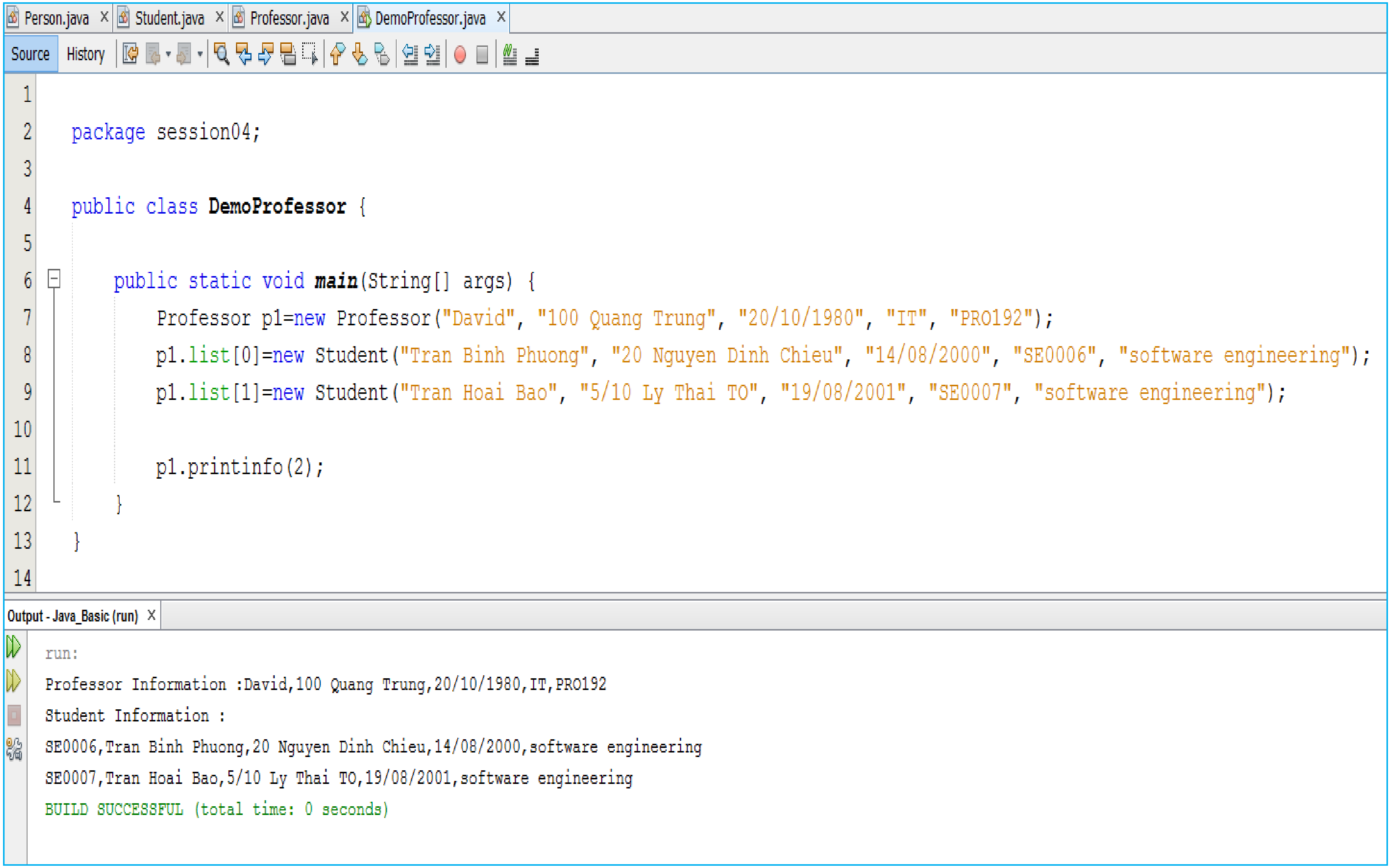
```java

package session04;

public class Student extends Person{
   String studentid;
   String majorfield;

    public Student(String name, String address, String birthday,String studentid, String majorfield) {
        super(name, address, birthday);
        this.studentid=studentid;
        this.majorfield=majorfield;
    }

    public String getStudentid() {...3 lines }

    public void setStudentid(String studentid) {...3 lines }

    public String getMajorfield() {...3 lines }

    public void setMajorfield(String majorfield) {...3 lines }

}

```

Demo Object-Oriented Relationships…



```

package session04;

public class Professor extends Person{
    String department;
    String subject;
    Student[] list =new Student[30];// danh sách sinh viên -> has a

    public Professor(String name, String address, String birthday,String department,String subject) {...6 lines }

    public String getDepartment() {...3 lines }

    public void setDepartment(String department) {...3 lines }

    public String getSubject() {...3 lines }

    public void setSubject(String subject) {...3 lines }

    // n là số sinh viên theo học giáo sư
    public void  printinfo(int n){
        System.out.println("Professor Information :" + name +","+ address+","+ birthday + "," + department +"," + subject);
        System.out.println("Student Information :");
        for(int i=0; i<n;i++){
            System.out.print(list[i].studentid +"," + list[i].name +","+ list[i].address + ",");
            System.out.println(list[i].birthday + "," + list[i].majorfield);

        }

    }

}
```

Demo Object-Oriented Relationships…



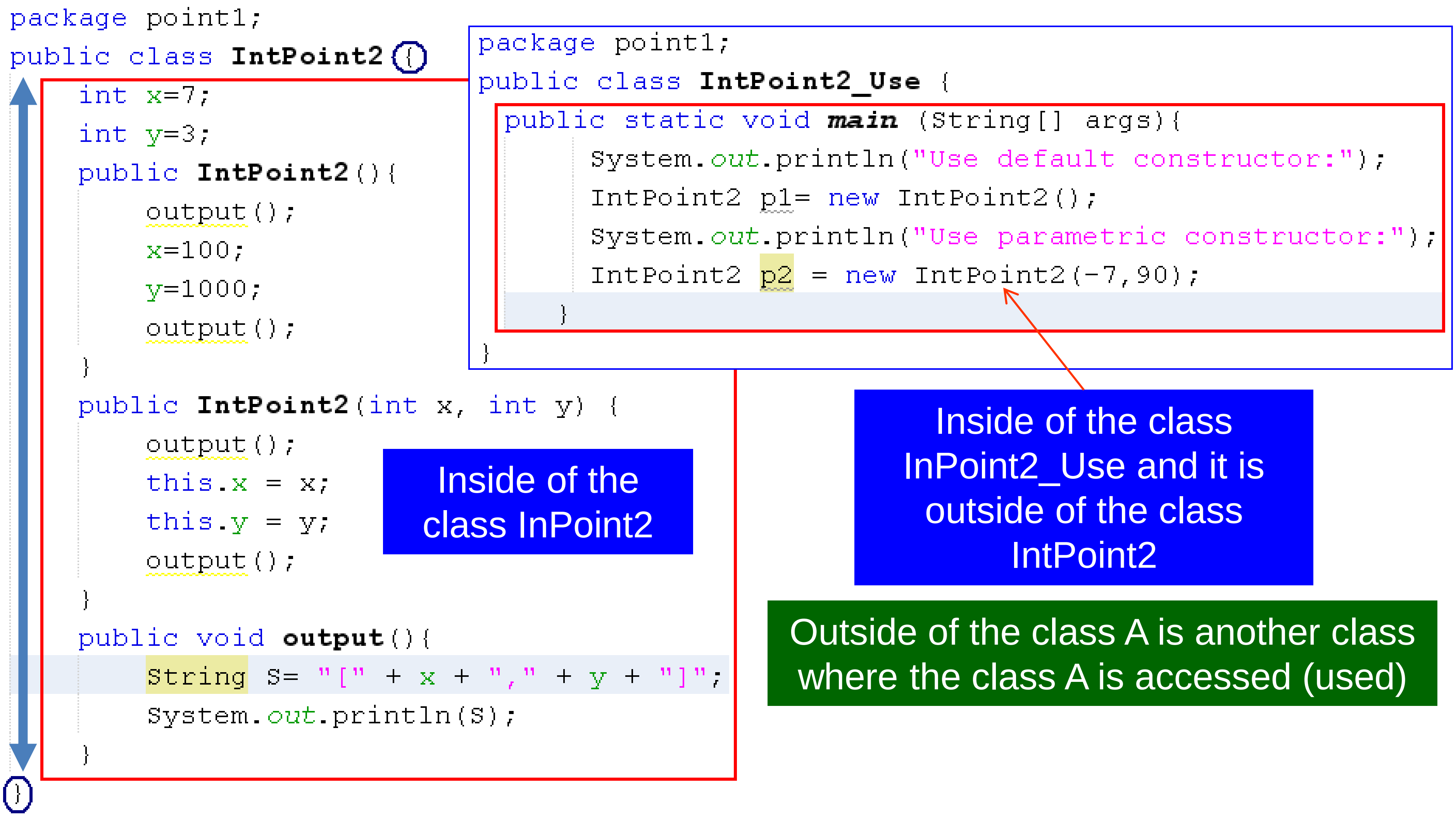
```java

package session04;

public class DemoProfessor {

    public static void main(String[] args) {
        Professor p1=new Professor("David", "100 Quang Trung", "20/10/1980", "IT", "PRO192");
        p1.list[0]=new Student("Tran Binh Phuong", "20 Nguyen Dinh Chieu", "14/08/2000", "SE0006", "software engineering");
        p1.list[1]=new Student("Tran Hoai Bao", "5/10 Ly Thai TO", "19/08/2001", "SE0007", "software engineering");

        p1.printinfo(2);
    }
}

```

```
run:
Professor Information :David,100 Quang Trung,20/10/1980,IT,PRO192
Student Information :
SE0006,Tran Binh Phuong,20 Nguyen Dinh Chieu,14/08/2000,software engineering
SE0007,Tran Hoai Bao,5/10 Ly Thai TO,19/08/2001,software engineering
BUILD SUCCESSFUL (total time: 0 seconds)
```

Common Modifiers

- Modifier (linguistics)is a word which can bring out the meaning of other word (adjective →noun, adverb → verb)
- Modifiers (OOP) are keywords that give the compiler information about the nature of code (methods), data, classes.
- Java supports some modifiers in which some of them are common and they are called as **access modifiers** (public, protected, default, private).
- Common modifiers will impose level of accessing on
  - class (where it can be used?)
  - methods (whether they can be called or not)
  - fields (whether they may be read/written or not)

Outside of a Class



```java
package point1;
public class IntPoint2 {
    int x=7;
    int y=3;
    public IntPoint2(){
        output();
        x=100;
        y=1000;
        output();
    }
    public IntPoint2(int x, int y) {
        output();
        this.x = x;
        this.y = y;
        output();
    }
    public void output(){
        String S= "[" + x + "," + y + "]";
        System.out.println(S);
    }
}
```

```java
package point1;
public class IntPoint2_Use {
  public static void main (String[] args){
        System.out.println("Use default constructor:");
        IntPoint2 p1= new IntPoint2();
        System.out.println("Use parametric constructor:");
        IntPoint2 p2 = new IntPoint2(-7,90);
     }
}
```

Inside of the class InPoint2

Inside of the class InPoint2_Use and it is outside of the class IntPoint2

Outside of the class A is another class where the class A is accessed (used)

Common Modifiers

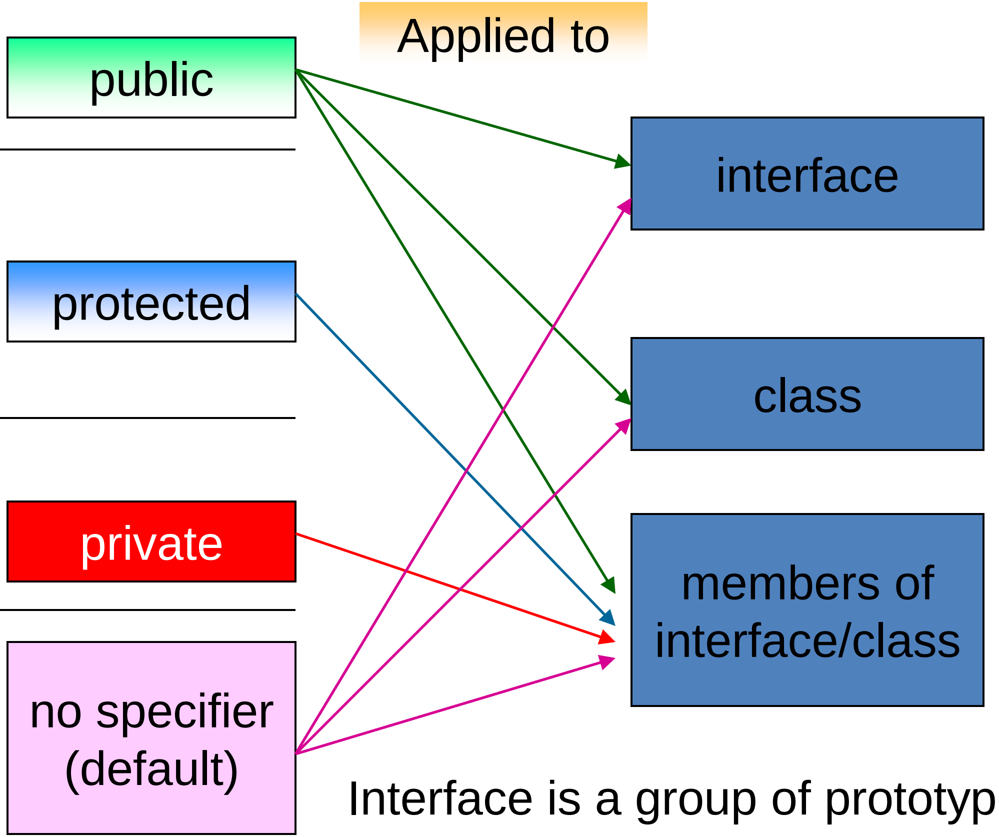
Access level

Free-accessing

package/ subclass

outside cannot access

package



Applied to

public

protected

private

no specifier (default)

interface

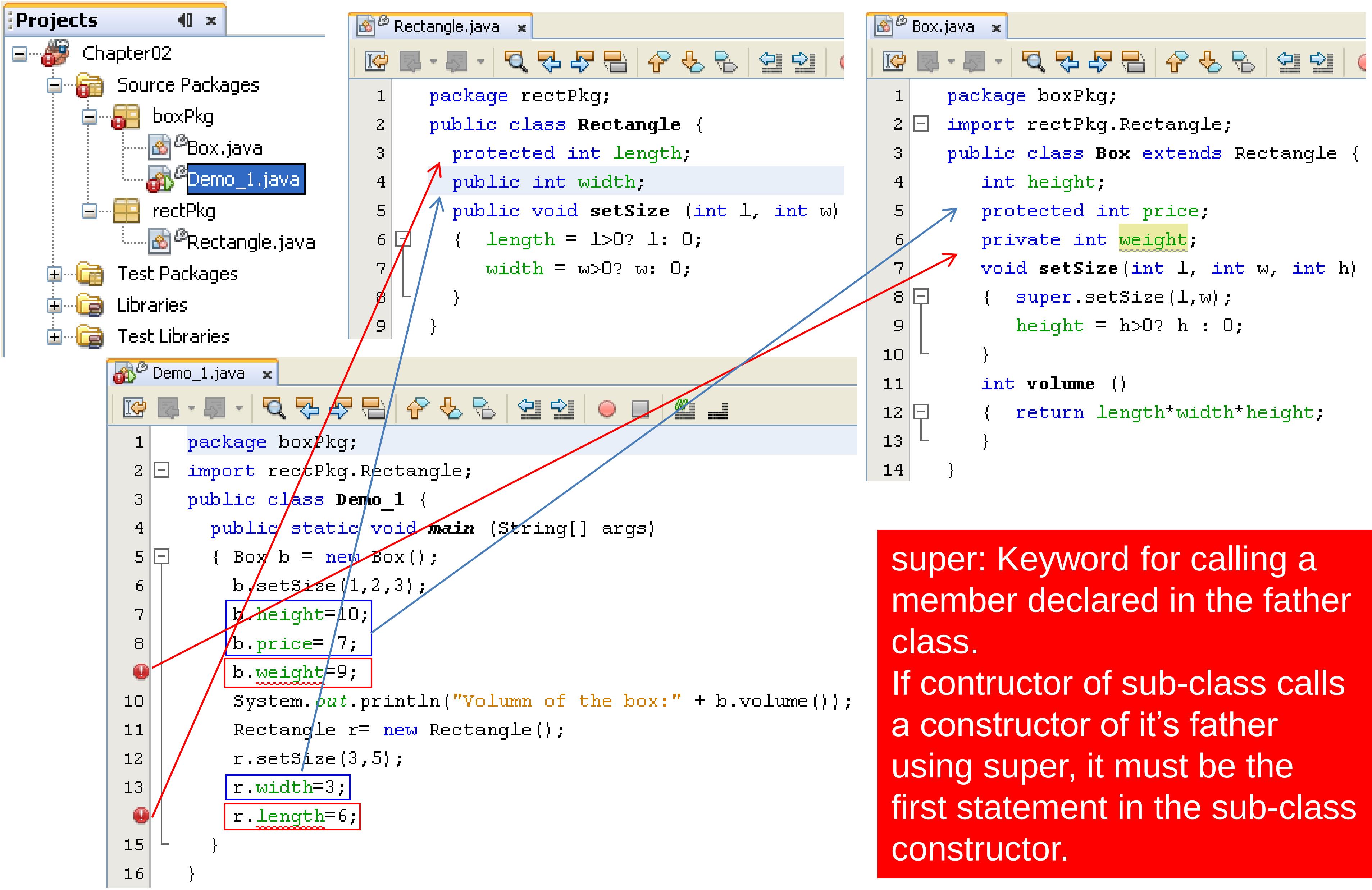
class

members of interface/class

Order:
public > protected > default > private

Interface is a group of prototyped methods and they will be implemented in a class afterward. It will be introduced later.

Common Modifiers



Rectangle.java

```java
package rectPkg;
public class Rectangle {
  protected int length;
  public int width;
  public void setSize (int l, int w)
  {  length = l>0? l: 0;
     width = w>0? w: 0;
  }
}
```

Box.java

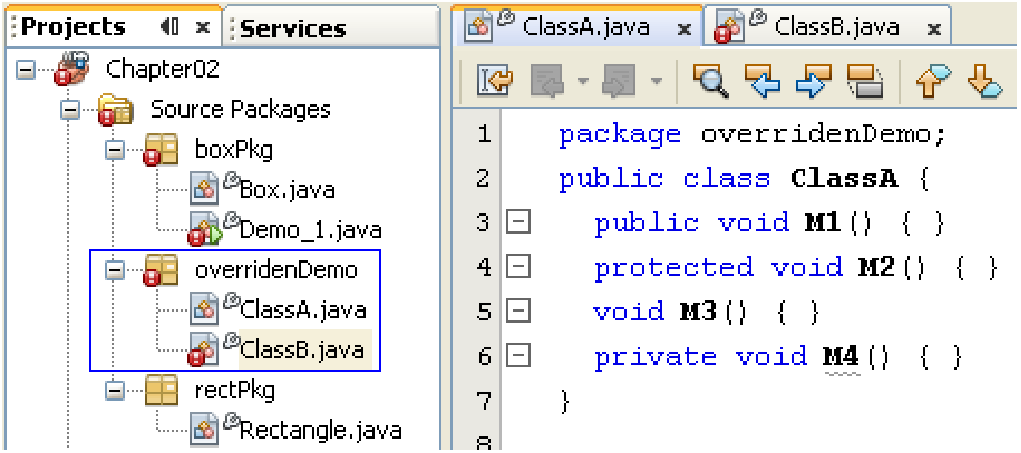
```java
package boxPkg;
import rectPkg.Rectangle;
public class Box extends Rectangle {
   int height;
   protected int price;
   private int weight;
   void setSize(int l, int w, int h)
   {  super.setSize(l,w);
      height = h>0? h : 0;
   }
   int volume ()
   {  return length*width*height;
   }
}
```

Demo_1.java

```java
package boxPkg;
import rectPkg.Rectangle;
public class Demo_1 {
  public static void main (String[] args)
  { Box b = new Box();
    b.setSize(1,2,3);
    b.height=10;
    b.price= 7;
    b.weight=9;
    System.out.println("Volumn of the box:" + b.volume());
    Rectangle r= new Rectangle();
    r.setSize(3,5);
    r.width=3;
    r.length=6;
  }
}
```

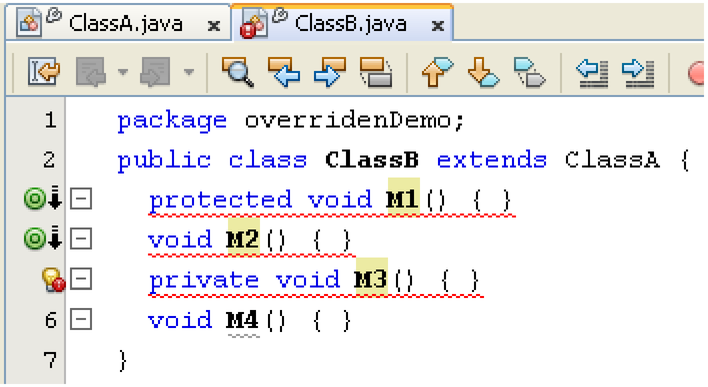
super: Keyword for calling a member declared in the father class.
If contructor of sub-class calls a constructor of it's father using super, it must be the first statement in the sub-class constructor.

Access Modifier Overridden

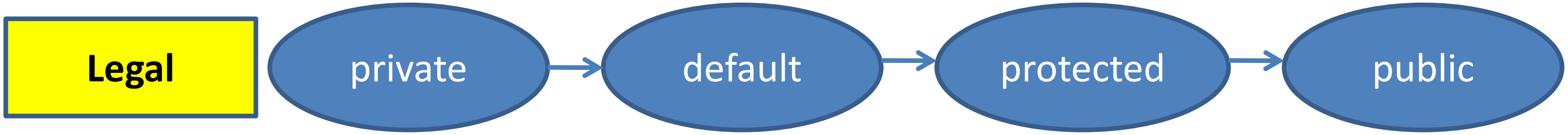


- Chapter02
  - Source Packages
    - boxPkg
      - Box.java
      - Demo_1.java
    - overridenDemo
      - ClassA.java
      - ClassB.java
    - rectPkg
      - Rectangle.java

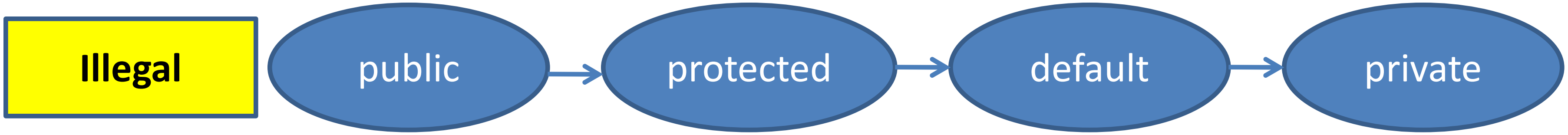
```java
package overridenDemo;
public class ClassA {
  public void M1() { }
  protected void M2() { }
  void M3() { }
  private void M4() { }
}
```



```java
package overridenDemo;
public class ClassB extends ClassA {
  protected void M1() { }
  void M2() { }
  private void M3() { }
  void M4() { }
}
```



Legal: private → default → protected → public

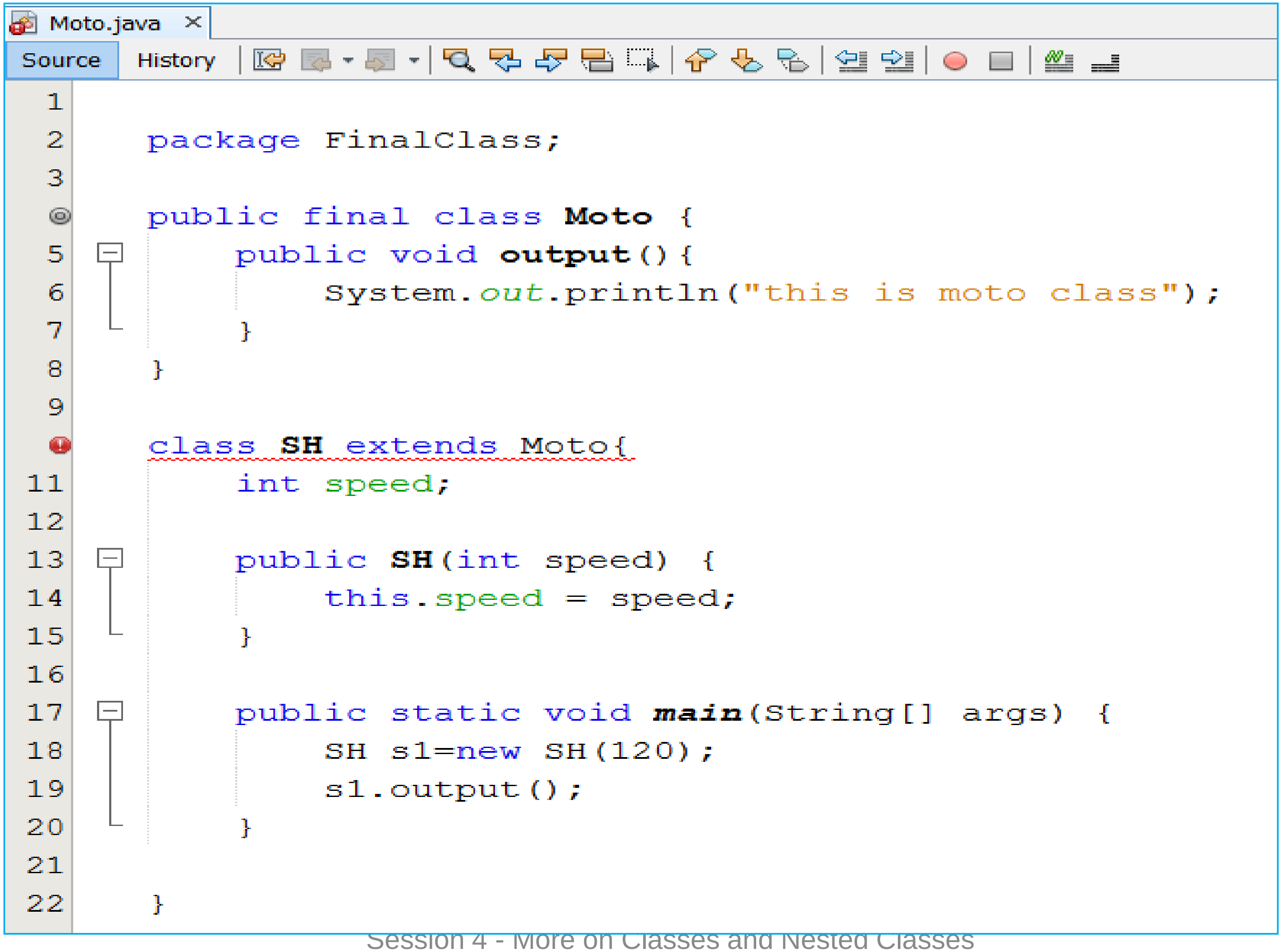


Illegal: public → protected → default → private

The sub-class must be more opened than it’s father

Modifier *final*

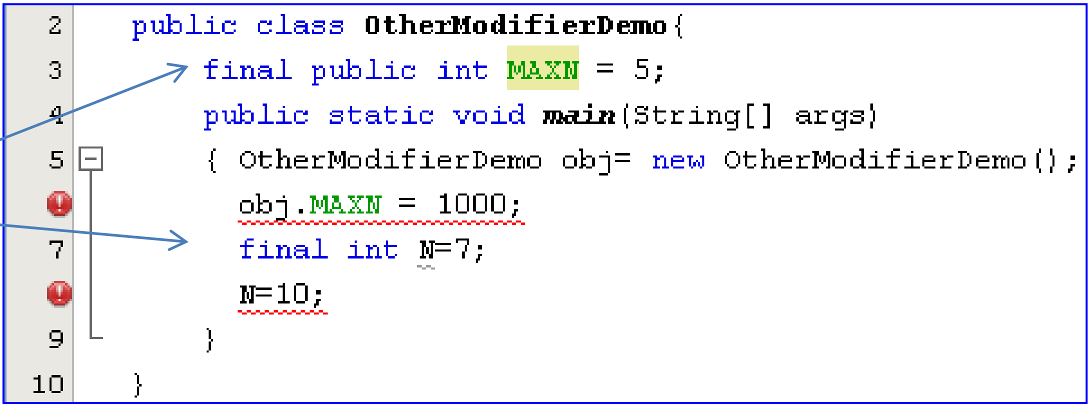
- Final class:Class can not have sub-class



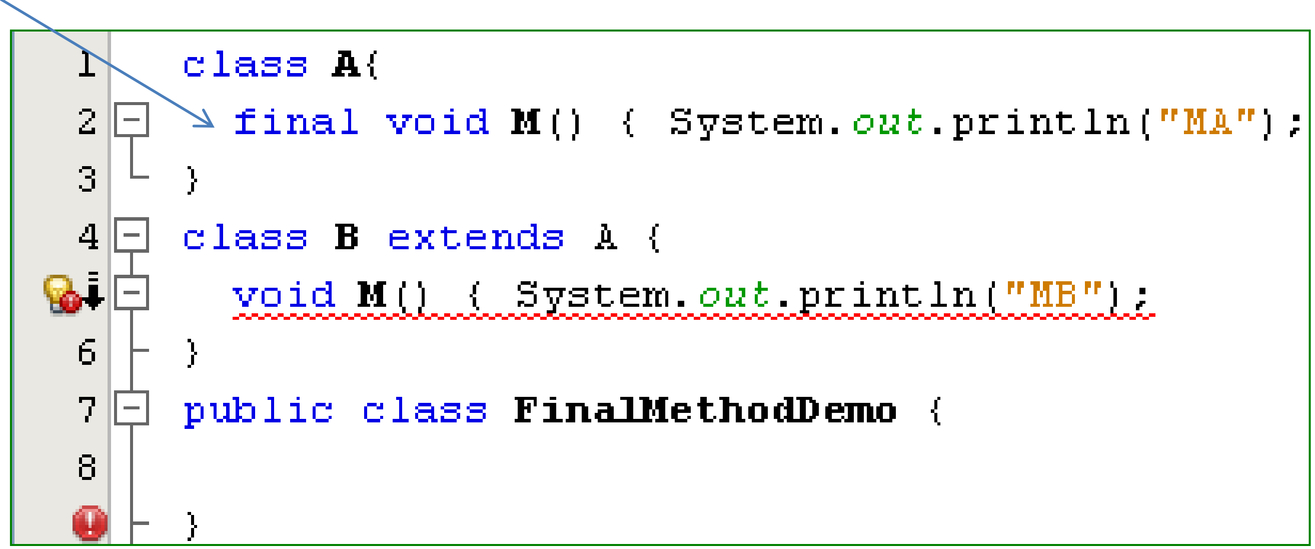
```

package FinalClass;

public final class Moto {
    public void output(){
        System.out.println("this is moto class");
    }
}

class SH extends Moto{
    int speed;

    public SH(int speed) {
        this.speed = speed;
    }

    public static void main(String[] args) {
        SH s1=new SH(120);
        s1.output();
    }

}
```

Modifier *final*

- Final data is a constant.
- Final method: a method can not be overridden.



```
public class OtherModifierDemo{
   final public int MAXN = 5;
   public static void main(String[] args)
   { OtherModifierDemo obj= new OtherModifierDemo();
     obj.MAXN = 1000;
     final int N=7;
     N=10;
   }
}
```



```
class A{
   final void M() { System.out.println("MA");
}
class B extends A {
   void M() { System.out.println("MB");
}
public class FinalMethodDemo {

}
```

Modifier *static*
Class variable/ Object variable

- Object variable: Variable of each object
- Class Variable: A variable is shared in all objects of class. It is stored separately. It is declared with the modifier ***static***
- Class variable is stored separately. So, it can be accessed as:

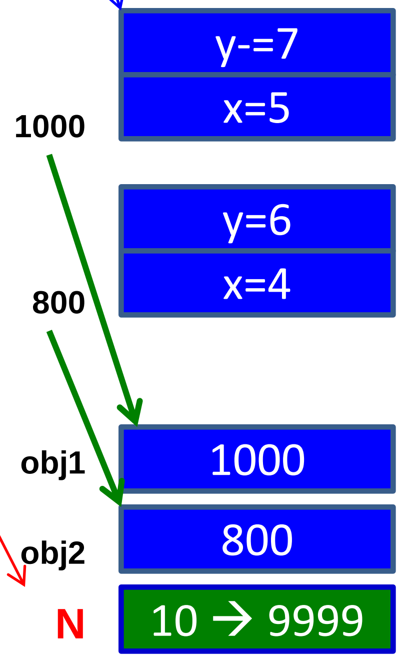
  object.staticVar

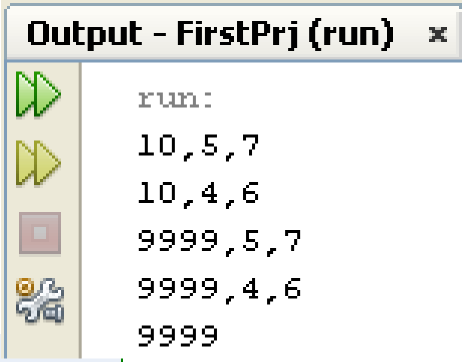
  ClassName.staticVar

Modifier *static*:Class variable/ Object variable

```
public class StaticVarDemo {
    static int N=10; // class variable
    int x, y; // object variable
    public StaticVarDemo(int xx, int yy){
        x= xx; y=yy;
    }
    public void setN( int nn){
        N= nn;
    }
    public void output(){
      System.out.println(N + "," + x + "," + y);
    }
}
```

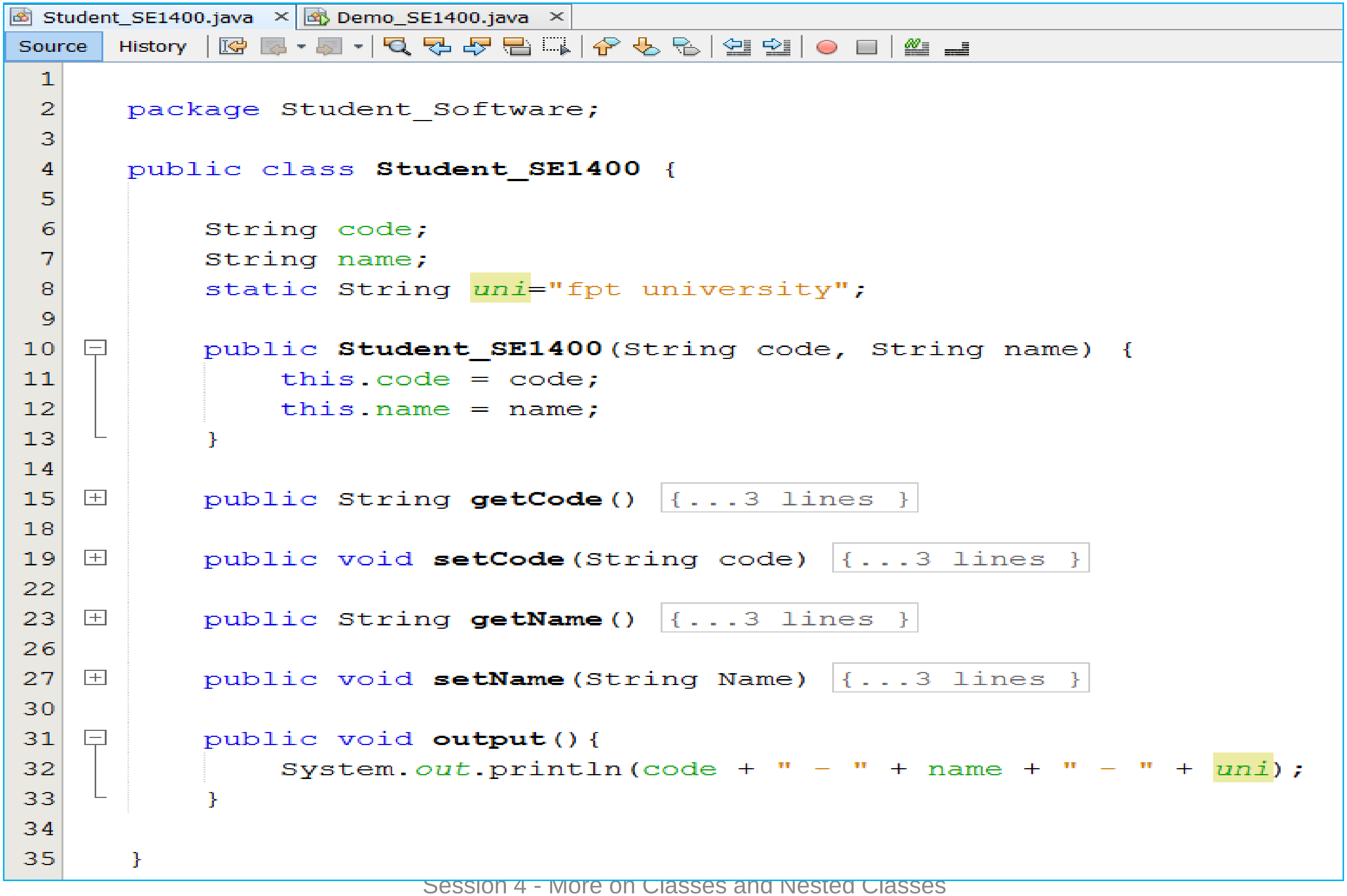
```
public class StaticVarDemoUse {
   public static void main(String args[]){
       StaticVarDemo obj1= new StaticVarDemo(5,7);
       StaticVarDemo obj2= new StaticVarDemo(4,6);
       obj1.output();
       obj2.output();
       obj1.setN(9999);
       obj1.output();
       obj2.output();
       System.out.println(StaticVarDemo.N);
   }
}
```





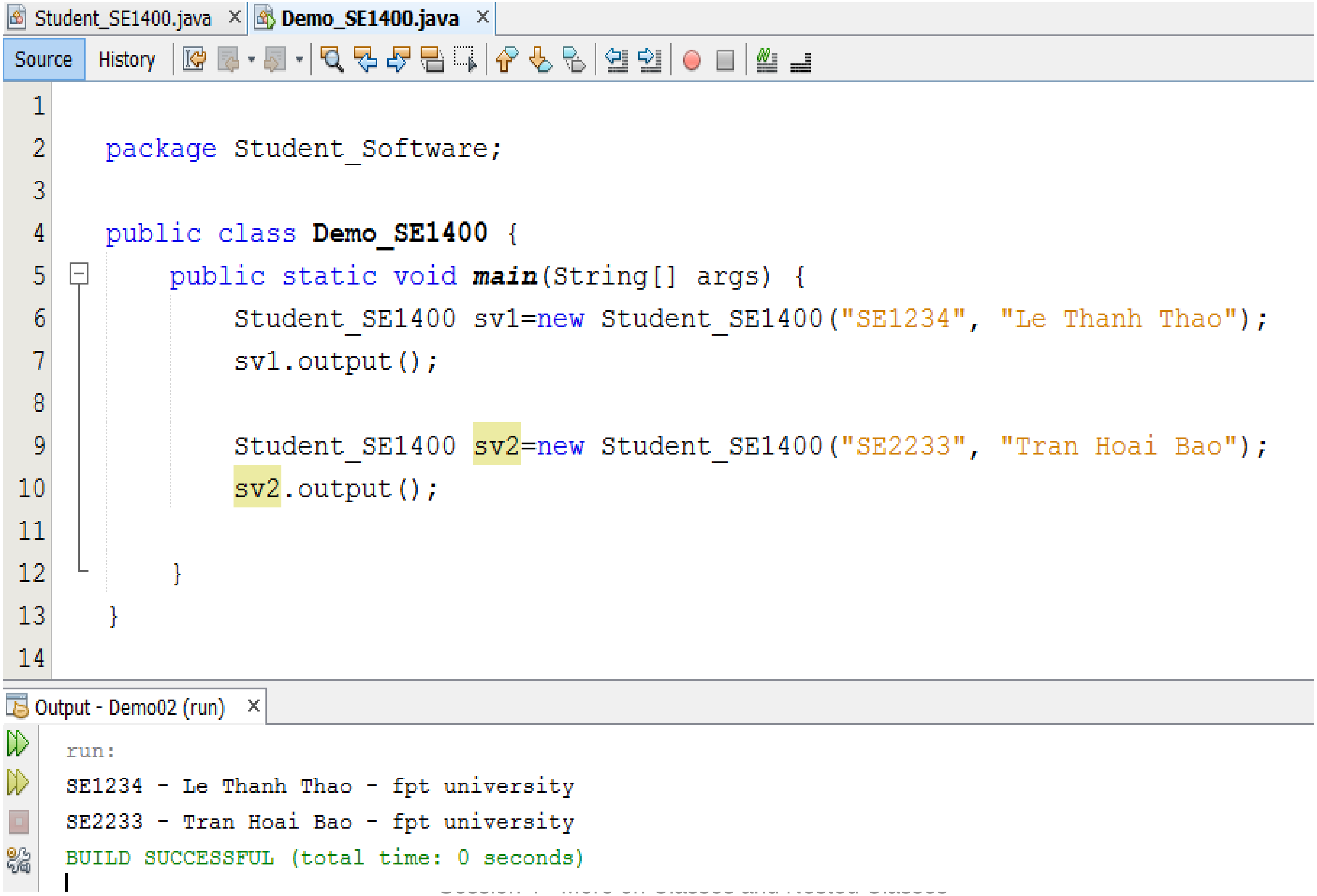
```
run:
10,5,7
10,4,6
9999,5,7
9999,4,6
9999
```

Example Modifier *static*



```
package Student_Software;

public class Student_SE1400 {

    String code;
    String name;
    static String uni="fpt university";

    public Student_SE1400(String code, String name) {
        this.code = code;
        this.name = name;
    }

    public String getCode() {...3 lines }

    public void setCode(String code) {...3 lines }

    public String getName() {...3 lines }

    public void setName(String Name) {...3 lines }

    public void output(){
        System.out.println(code + " - " + name + " - " + uni);
    }

}
```

Modifier *static*:



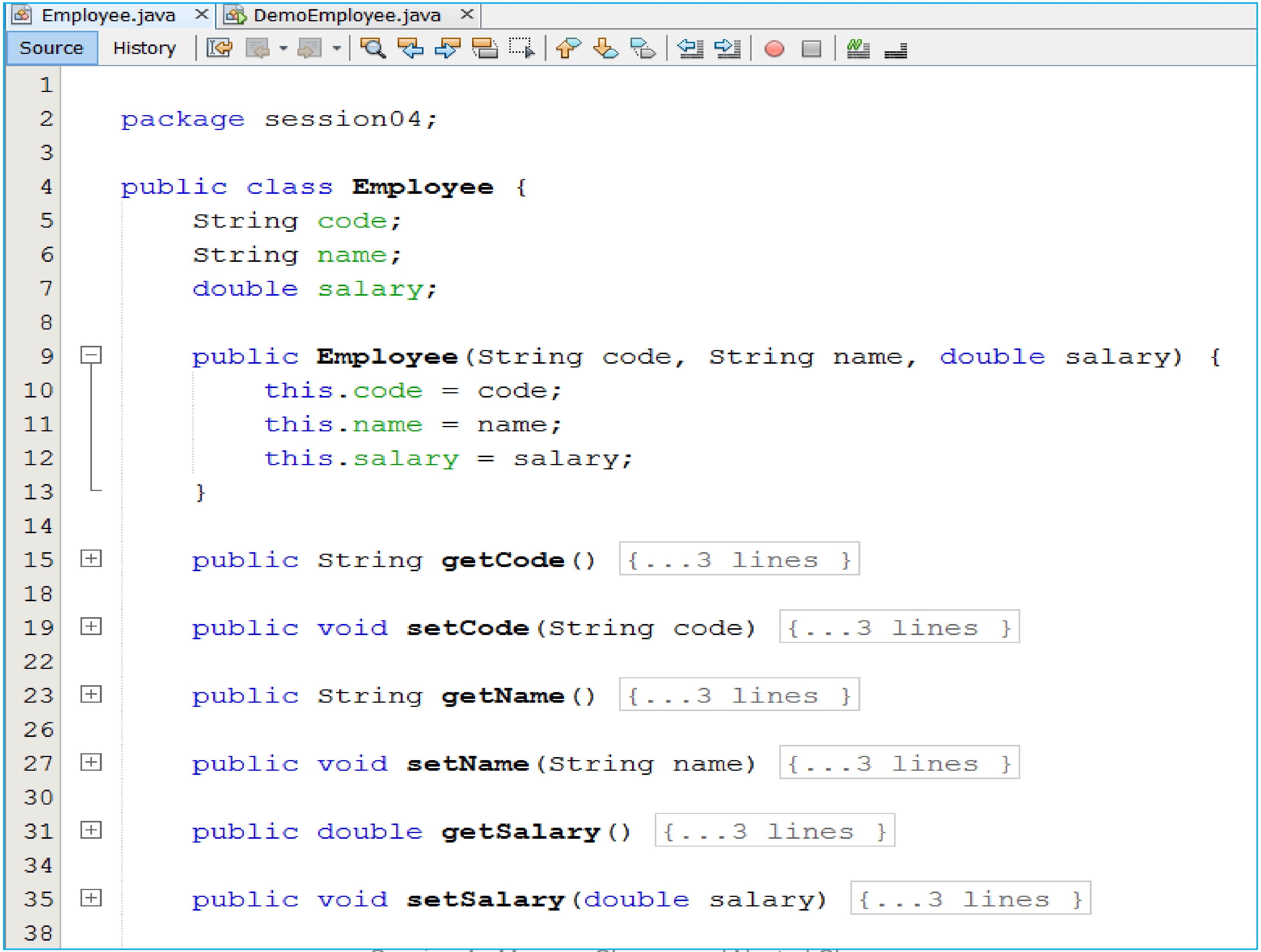
```java

package Student_Software;

public class Demo_SE1400 {
    public static void main(String[] args) {
        Student_SE1400 sv1=new Student_SE1400("SE1234", "Le Thanh Thao");
        sv1.output();

        Student_SE1400 sv2=new Student_SE1400("SE2233", "Tran Hoai Bao");
        sv2.output();

    }
}

```

```
run:
SE1234 - Le Thanh Thao - fpt university
SE2233 - Tran Hoai Bao - fpt university
BUILD SUCCESSFUL (total time: 0 seconds)
```

Modifier *static*: Static method

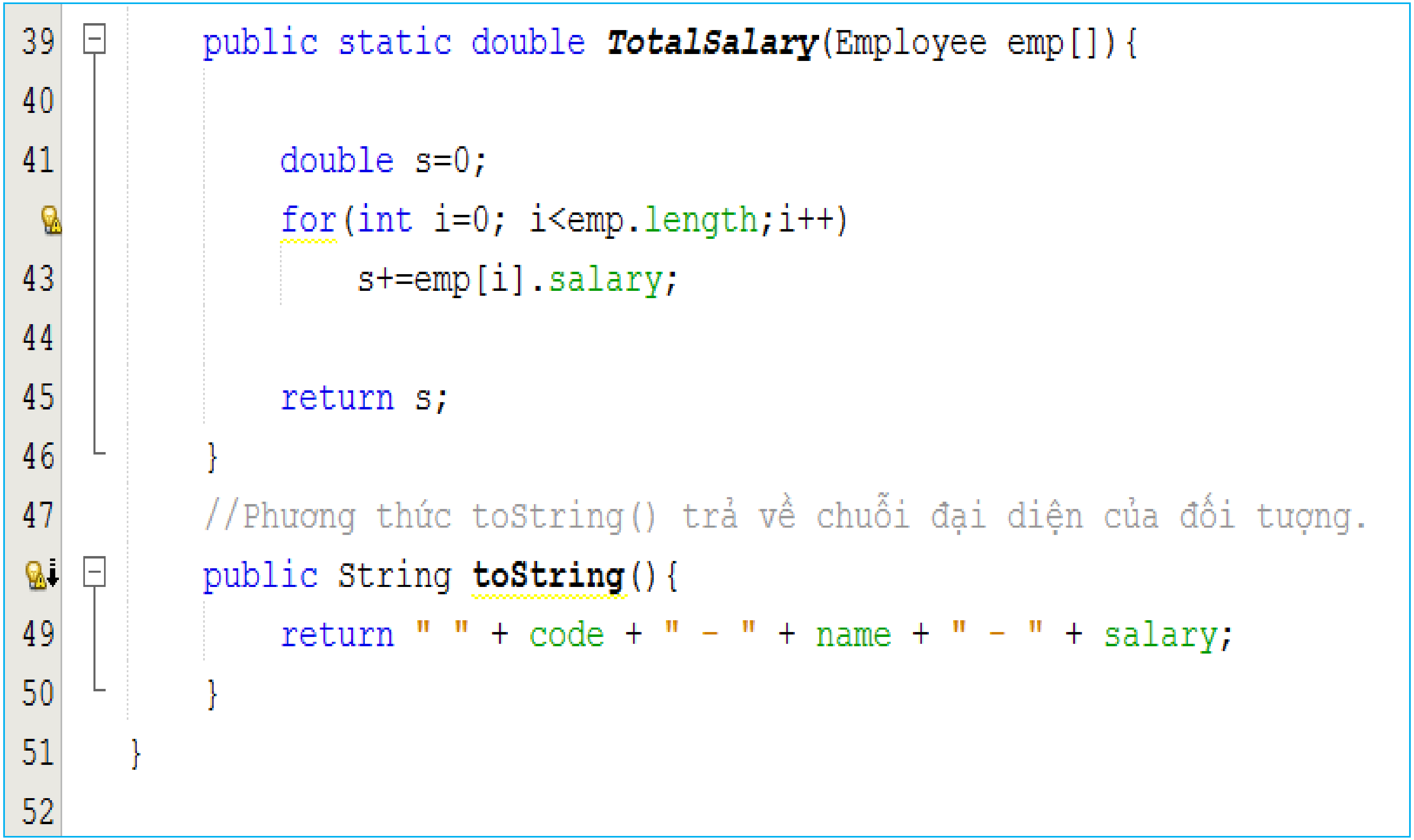
- It is called as class method/global method and it is called although no object of this class is created.
- Entry point of Java program is a static method
- Syntax for calling it: ClassName.staticMethod(args)
- ***Static methods***:
  - can access class variables and class methods directly **only**.
  - ***cannot*** access instance variables or instance methods directly—they must use an object reference.
  - cannot use the this keyword as there is no instance for this to refer to.

Modifier *static*: Static method



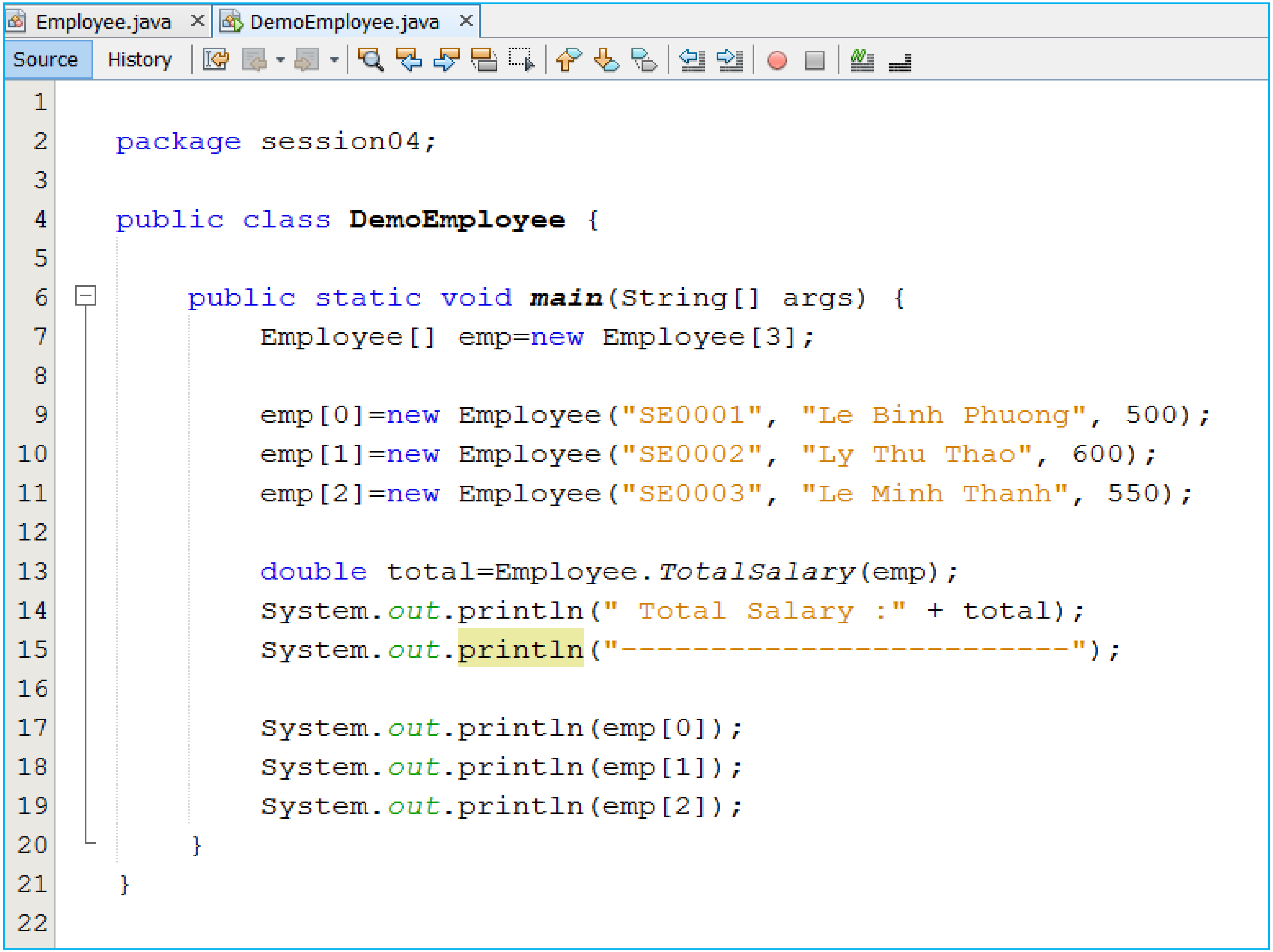
```java

package session04;

public class Employee {
    String code;
    String name;
    double salary;

    public Employee(String code, String name, double salary) {
        this.code = code;
        this.name = name;
        this.salary = salary;
    }

    public String getCode() {...3 lines }

    public void setCode(String code) {...3 lines }

    public String getName() {...3 lines }

    public void setName(String name) {...3 lines }

    public double getSalary() {...3 lines }

    public void setSalary(double salary) {...3 lines }

```

Modifier *static*: Static method



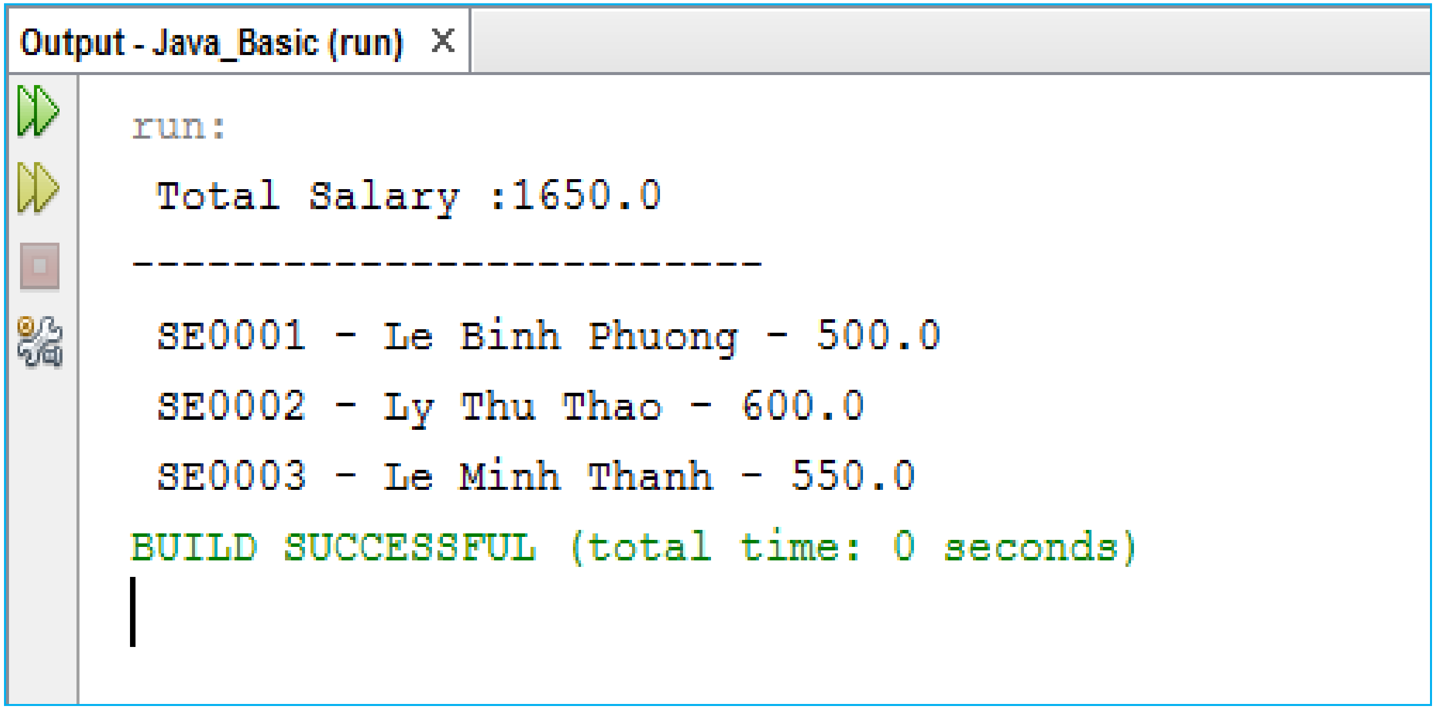
```
    public static double TotalSalary(Employee emp[]){

        double s=0;
        for(int i=0; i<emp.length;i++)
            s+=emp[i].salary;

        return s;
    }
    //Phương thức toString() trả về chuỗi đại diện của đối tượng.
    public String toString(){
        return " " + code + " - " + name + " - " + salary;
    }
}

```

Modifier *static*: Static method



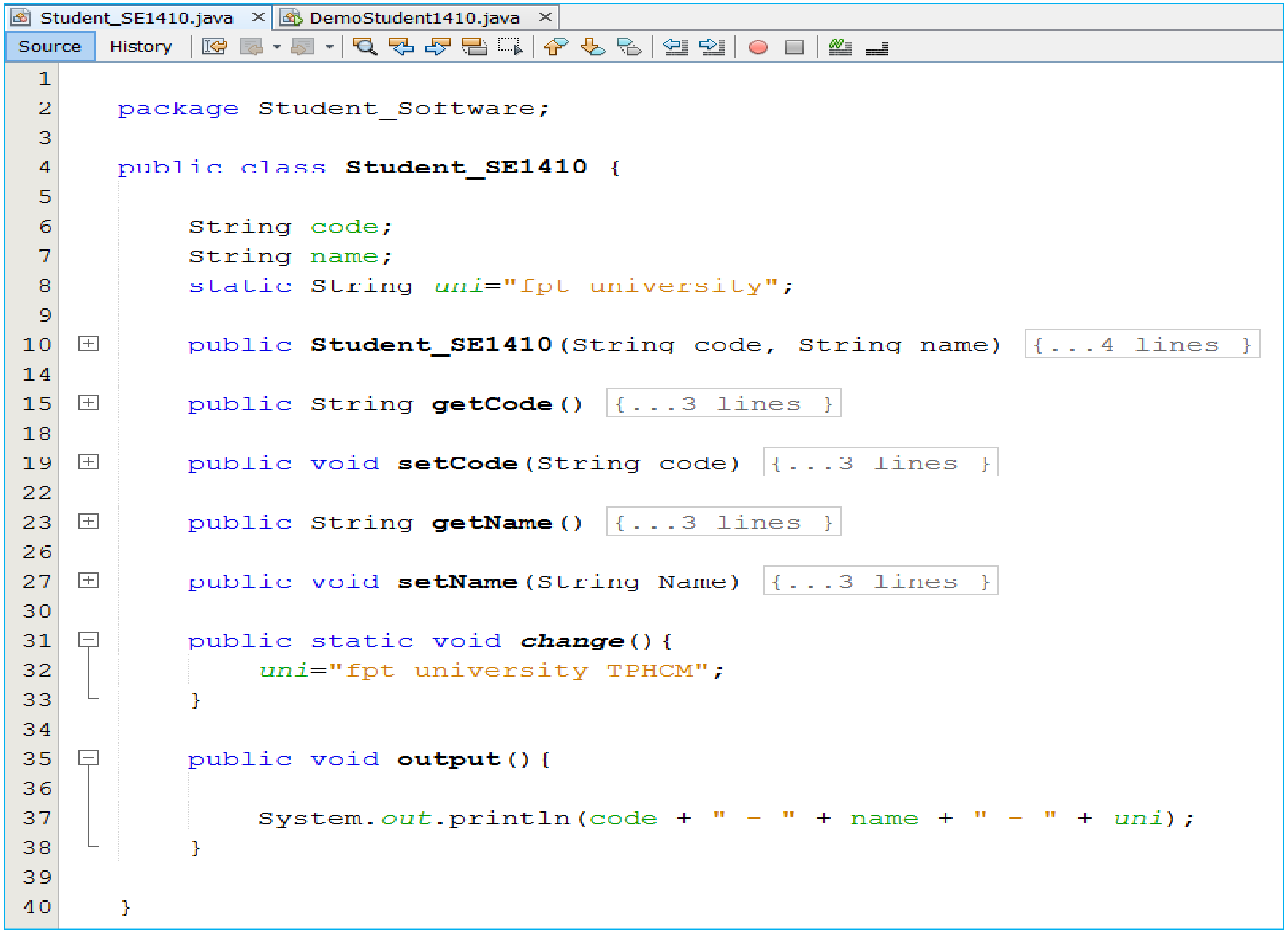
```java

package session04;

public class DemoEmployee {

    public static void main(String[] args) {
        Employee[] emp=new Employee[3];

        emp[0]=new Employee("SE0001", "Le Binh Phuong", 500);
        emp[1]=new Employee("SE0002", "Ly Thu Thao", 600);
        emp[2]=new Employee("SE0003", "Le Minh Thanh", 550);

        double total=Employee.TotalSalary(emp);
        System.out.println(" Total Salary :" + total);
        System.out.println("-------------------------");

        System.out.println(emp[0]);
        System.out.println(emp[1]);
        System.out.println(emp[2]);
    }
}

```

Modifier *static*: Static method



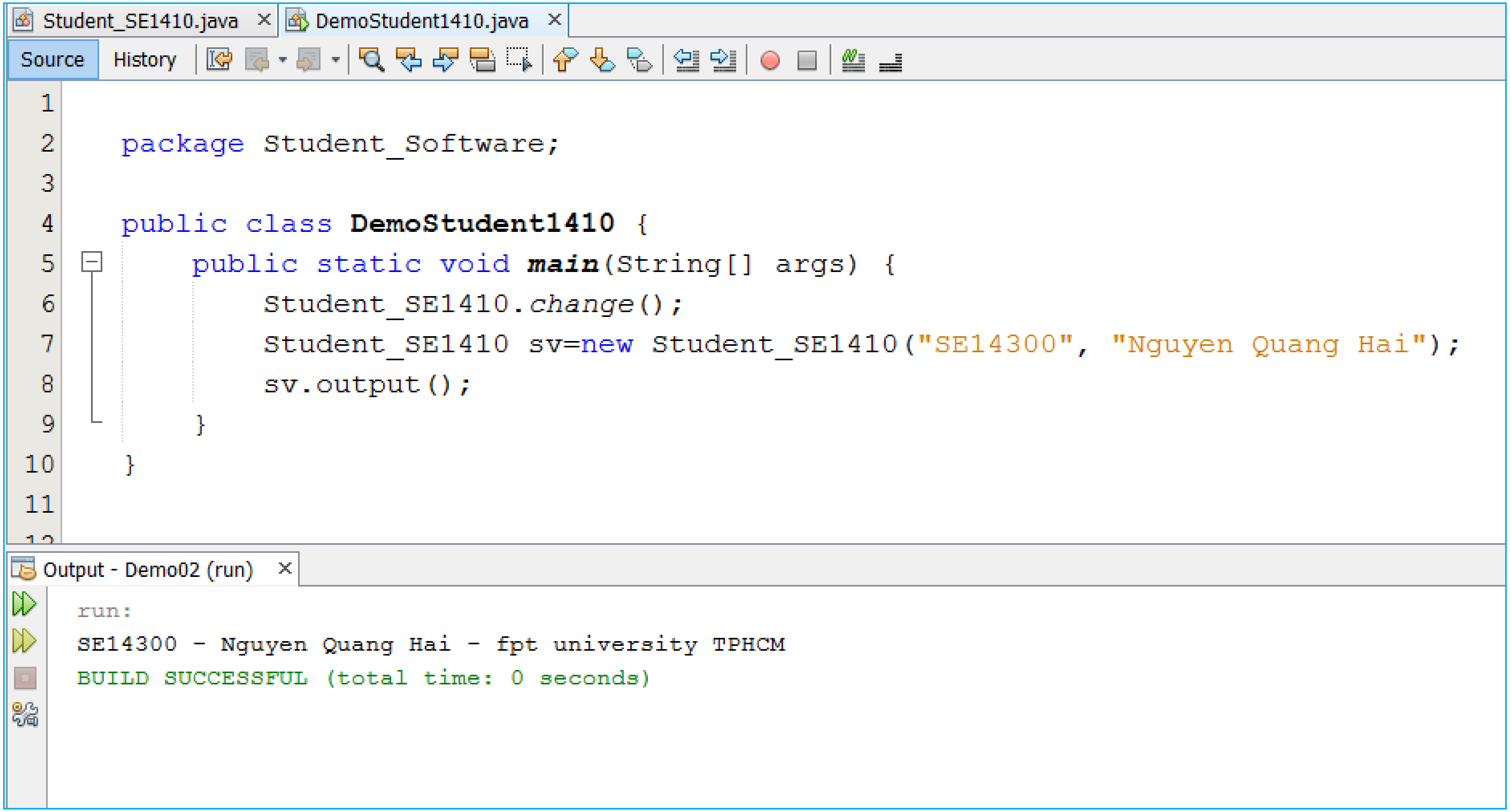
```
run:
 Total Salary :1650.0
-------------------------
 SE0001 - Le Binh Phuong - 500.0
 SE0002 - Ly Thu Thao - 600.0
 SE0003 - Le Minh Thanh - 550.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Static methods can access static variables and can change their values.



```
package Student_Software;

public class Student_SE1410 {

    String code;
    String name;
    static String uni="fpt university";

    public Student_SE1410(String code, String name) {...4 lines }

    public String getCode() {...3 lines }

    public void setCode(String code) {...3 lines }

    public String getName() {...3 lines }

    public void setName(String Name) {...3 lines }

    public static void change(){
        uni="fpt university TPHCM";
    }

    public void output(){

        System.out.println(code + " - " + name + " - " + uni);
    }

}
```

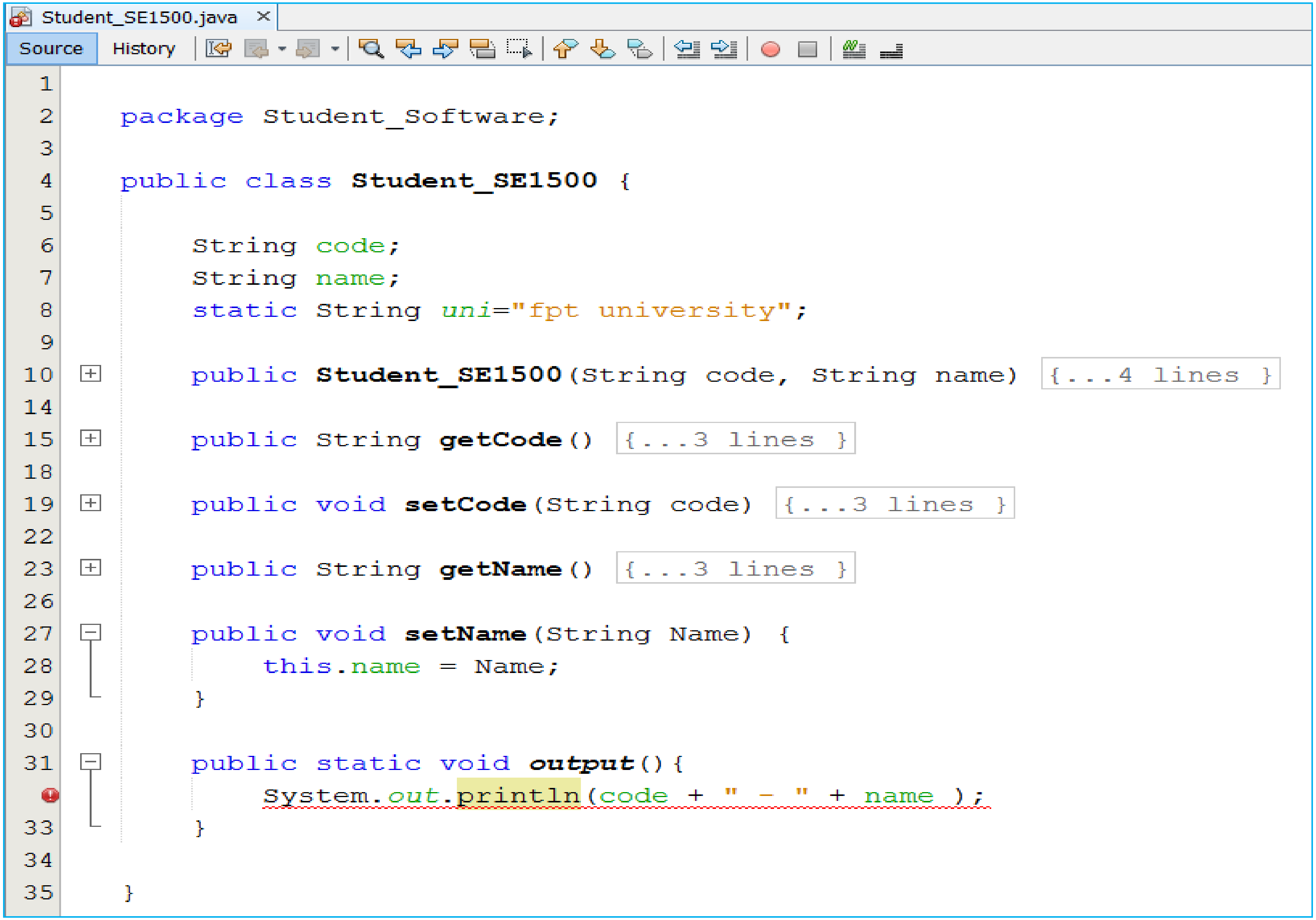
Static methods can access static variables and can change their values.



```java

package Student_Software;

public class DemoStudent1410 {
    public static void main(String[] args) {
        Student_SE1410.change();
        Student_SE1410 sv=new Student_SE1410("SE14300", "Nguyen Quang Hai");
        sv.output();
    }
}

```

```
run:
SE14300 - Nguyen Quang Hai - fpt university TPHCM
BUILD SUCCESSFUL (total time: 0 seconds)
```

Modifier *static*: Static method



```
package Student_Software;

public class Student_SE1500 {

    String code;
    String name;
    static String uni="fpt university";

    public Student_SE1500(String code, String name) {...4 lines }

    public String getCode() {...3 lines }

    public void setCode(String code) {...3 lines }

    public String getName() {...3 lines }

    public void setName(String Name) {
        this.name = Name;
    }

    public static void output(){
        System.out.println(code + " - " + name );
    }

}
```

Modifier *static*:
What should be static in a class?

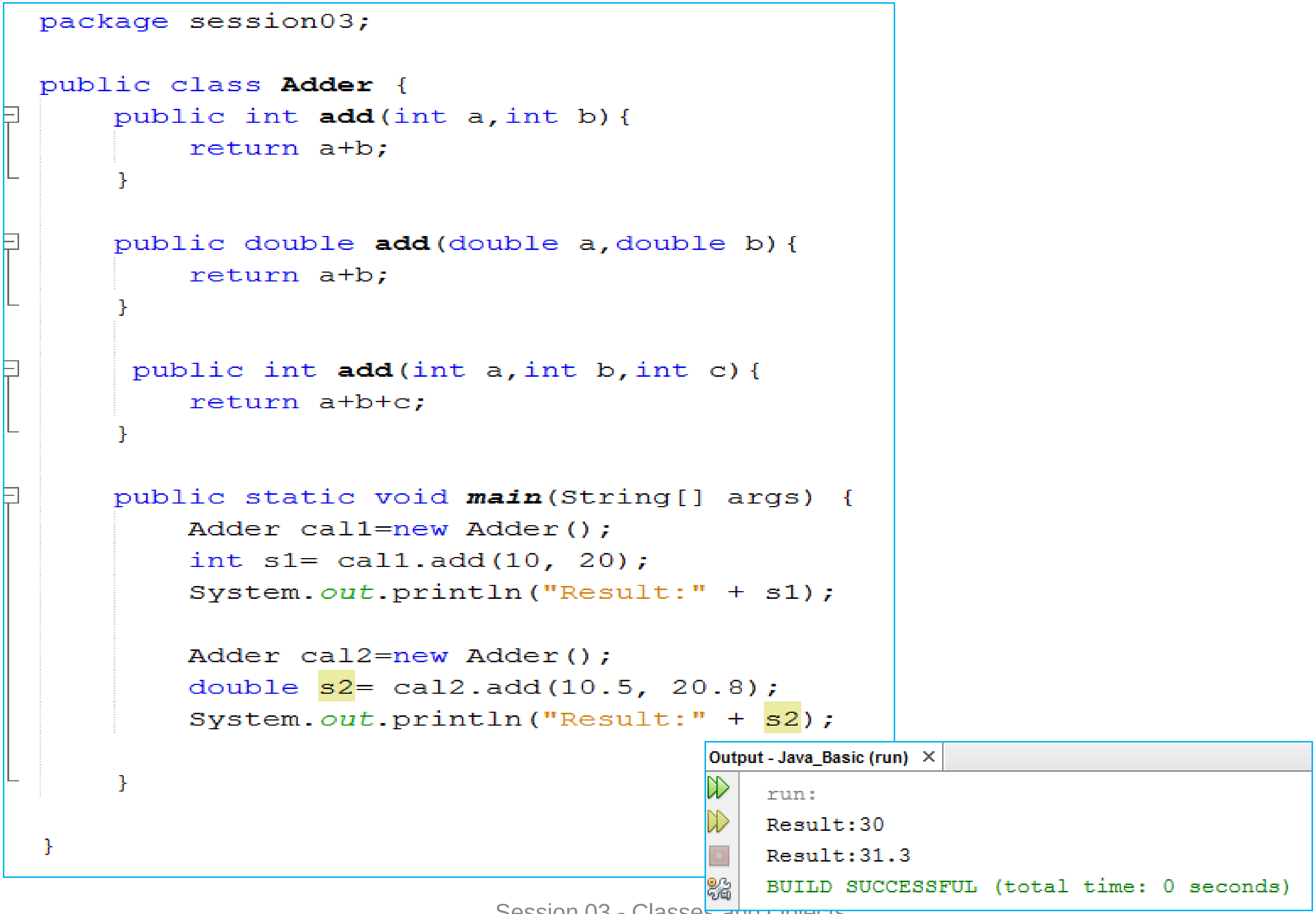
- Constants:
  - The static modifier, in combination with the final modifier, is also used to define constants. The final modifier indicates that the value of this field cannot change.

```
static final double PI = 3.141592653589793;
```

Overloading Method

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

Overloading Method



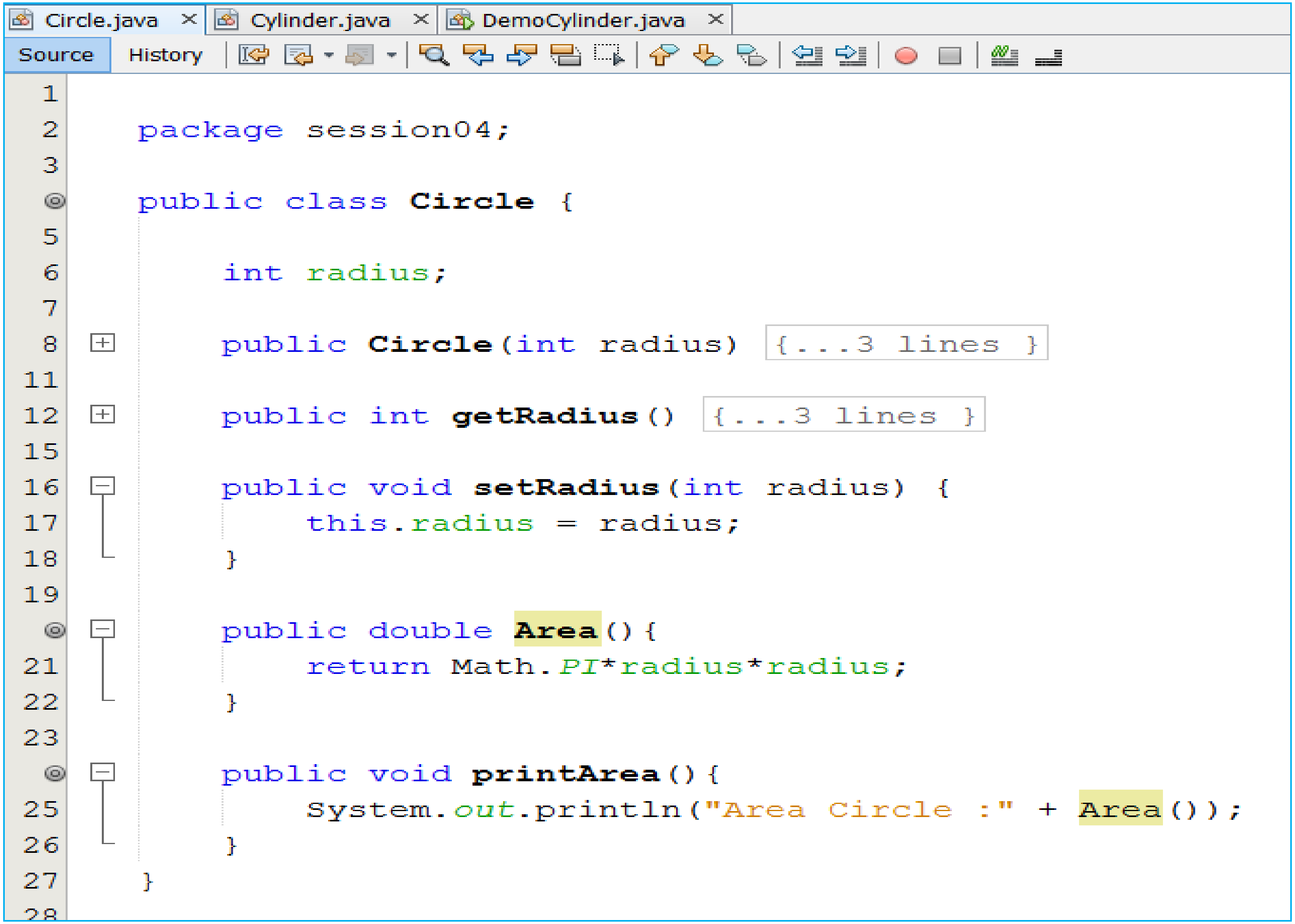
```java
package session03;

public class Adder {
    public int add(int a,int b){
        return a+b;
    }

    public double add(double a,double b){
        return a+b;
    }

     public int add(int a,int b,int c){
        return a+b+c;
    }

    public static void main(String[] args) {
        Adder cal1=new Adder();
        int s1= cal1.add(10, 20);
        System.out.println("Result:" + s1);

        Adder cal2=new Adder();
        double s2= cal2.add(10.5, 20.8);
        System.out.println("Result:" + s2);

    }

}
```

```
run:
Result:30
Result:31.3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Overriding Methods

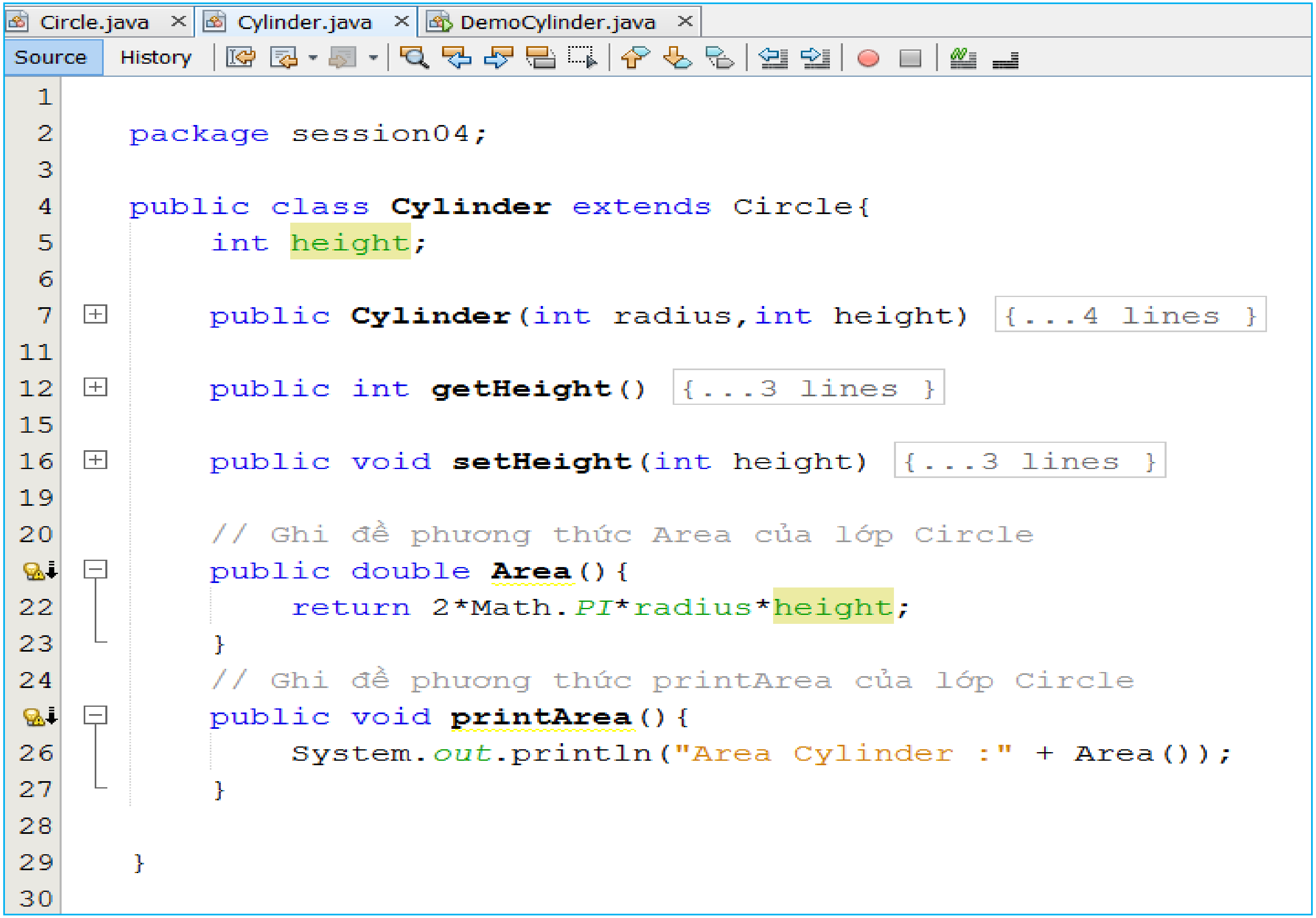
- **Overriding a method**: An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method.
  - Use the **@Override** annotation that instructs the compiler that you intend to override a method in the superclass (you may not use it because overriding is the default in Java).
- **Hiding a method**: Re-implementing a static method implemented in super class

Demo Overriding



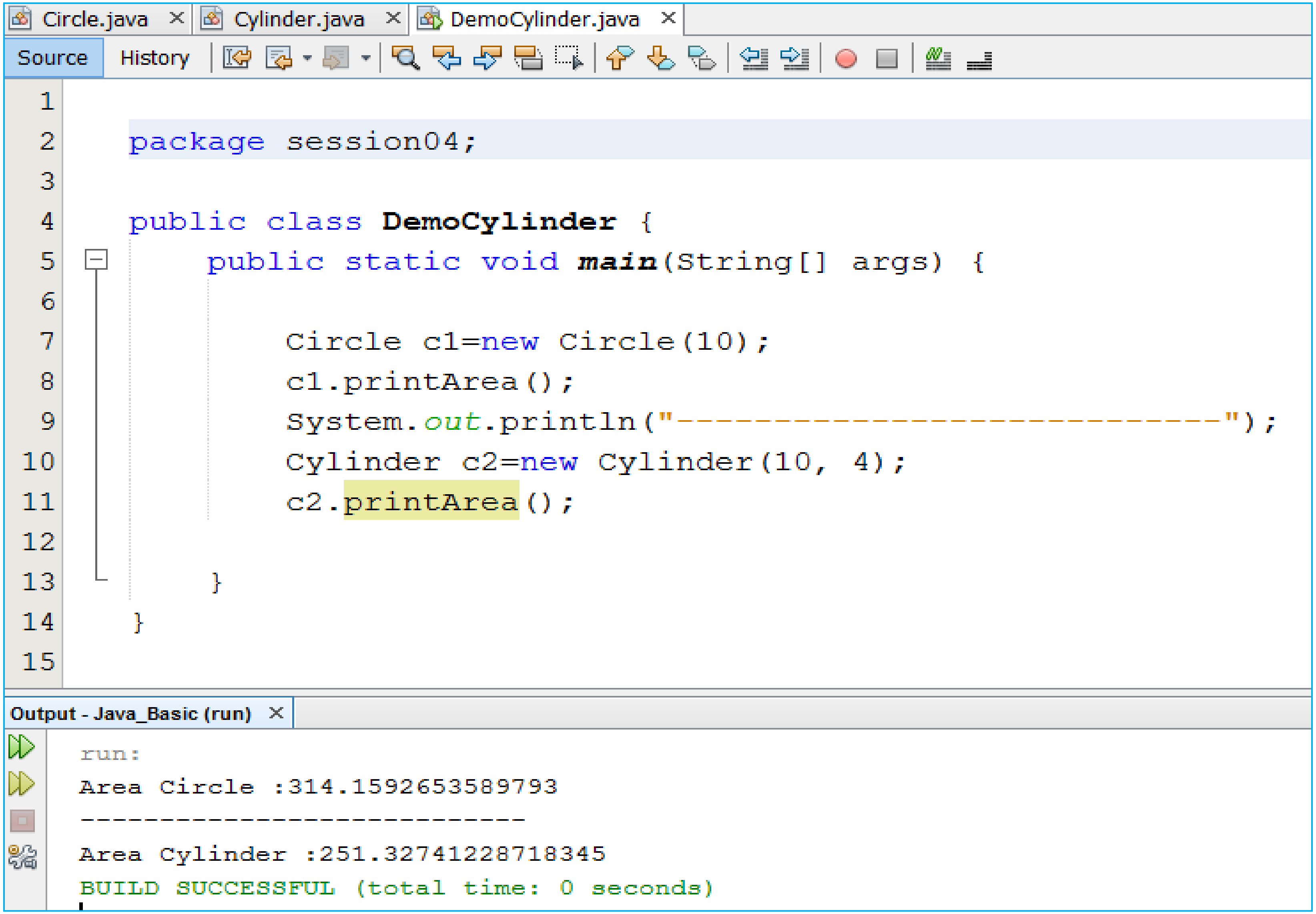
```
package session04;

public class Circle {

    int radius;

    public Circle(int radius) {...3 lines }

    public int getRadius() {...3 lines }

    public void setRadius(int radius) {
        this.radius = radius;
    }

    public double Area(){
        return Math.PI*radius*radius;
    }

    public void printArea(){
        System.out.println("Area Circle :" + Area());
    }
}
```

Demo Overriding



```
package session04;

public class Cylinder extends Circle{
    int height;

    public Cylinder(int radius,int height) {...4 lines }

    public int getHeight() {...3 lines }

    public void setHeight(int height) {...3 lines }

    // Ghi đề phương thức Area của lớp Circle
    public double Area(){
        return 2*Math.PI*radius*height;
    }
    // Ghi đề phương thức printArea của lớp Circle
    public void printArea(){
        System.out.println("Area Cylinder :" + Area());
    }

}
```

Demo Overriding



```java
package session04;

public class DemoCylinder {
    public static void main(String[] args) {

        Circle c1=new Circle(10);
        c1.printArea();
        System.out.println("------------------------------");
        Cylinder c2=new Cylinder(10, 4);
        c2.printArea();

    }
}
```

```
run:
Area Circle :314.1592653589793
------------------------------
Area Cylinder :251.32741228718345
BUILD SUCCESSFUL (total time: 0 seconds)
```

Summary

- Object-oriented languages implement reusability of coding structure through inheritance
- A derived class does not by default inherit the constructor of a super class
- Constructors in an inheritance hierarchy execute in order from the super class to the derived class
- Using the instanceof keyword if we need to check the type of the reference variable.
- Check the type of the reference variable before casting it explicitly.